

07. 2. 2005

日本国特許庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されて
いる事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed
with this Office.

出願年月日 2004年 2月12日
Date of Application:

出願番号 特願2004-035430
Application Number:
[ST. 10/C]: [JP2004-035430]

出願人 松下電器産業株式会社
Applicant(s):

2005年 3月18日

特許長官
Commissioner,
Japan Patent Office

小川洋

BEST AVAILABLE COPY

出証番号 出証特2005-3024182

【書類名】 特許願
【整理番号】 5037750101
【あて先】 特許庁長官殿
【国際特許分類】 G06F 9/45
【発明者】
【住所又は居所】 大阪府門真市大字門真1006番地 松下電器産業株式会社内
【氏名】 川端 輝雄
【発明者】
【住所又は居所】 大阪府門真市大字門真1006番地 松下電器産業株式会社内
【氏名】 小川 一
【発明者】
【住所又は居所】 大阪府門真市大字門真1006番地 松下電器産業株式会社内
【氏名】 瓶子 岳人
【発明者】
【住所又は居所】 大阪府門真市大字門真1006番地 松下電器産業株式会社内
【氏名】 山本 康博
【特許出願人】
【識別番号】 000005821
【氏名又は名称】 松下電器産業株式会社
【代理人】
【識別番号】 100109210
【弁理士】
【氏名又は名称】 新居 広守
【手数料の表示】
【予納台帳番号】 049515
【納付金額】 21,000円
【提出物件の目録】
【物件名】 特許請求の範囲 1
【物件名】 明細書 1
【物件名】 図面 1
【物件名】 要約書 1
【包括委任状番号】 0213583

【書類名】特許請求の範囲**【請求項 1】**

実行時にインターロックを起こす可能性のある命令を備えたプロセッサ向けのコンパイラであって、

入力プログラムに対し、ループ回数が x 回のループをループ回数が y 回のループに分割し、前記ループ回数が y 回のループを内側ループとし、ループ回数が x / y 回のループを外側ループとする二重ループ変換を行うループ構造変換手段と、

前記二重ループ変換後のプログラムに対して、インターロックを起こす可能性のある命令の配置を行う命令最適配置手段としてコンピュータを機能させることを特徴とするコンパイラ。

【請求項 2】

さらに、最適化に関する最適化指示情報を受け取る最適化指定情報付加手段としてコンピュータを機能させることを特徴とする請求項 1 記載のコンパイラ。

【請求項 3】

前記命令はプリフェッチ命令である

ことを特徴とする請求項 1 または 2 記載のコンパイラ

【請求項 4】

前記ループ構造変換手段は、

前記ループ回数が x 回のループを当該ループ内で参照される配列のアドレスがキャッシュのラインサイズ進む分だけ実行されるループ回数が y 回のループに分割し、ループ回数が y 回のループを内側ループ、ループ回数が x / y 回のループを外側とする二重ループ変換を行う

ことを特徴とする請求項 1 ~ 3 のいずれか 1 項に記載のコンパイラ。

【請求項 5】

さらに、命令のスケジューリングを行うスケジューリング手段としてコンピュータを機能させ、

前記ループ構造変換手段は、

ループ回数が x 回のループを前記スケジューリングにより得られた結果から、プリフェッチにかかるサイクル数分だけ実行されるループ回数が y 回のループに分割し、ループ回数が y 回のループを内側ループ、ループ回数が x / y 回のループを外側とする二重ループ変換を行う

ことを特徴とする請求項 1 ~ 3 のいずれか 1 項に記載のコンパイラ。

【請求項 6】

前記ループ構造変換手段は、前記配列が複数存在する場合に、前記二重ループ変換を行った前記ループ回数が y 回のループをさらに、前記配列の存在数だけ配分する配分変換を行う

ことを特徴とする請求項 4 または 5 に記載のコンパイラ。

【請求項 7】

前記配分変換は、複数の前記配列について、その配列要素のサイズが異なるときに、前記サイズ比に応じて配分する

ことを特徴とする請求項 6 に記載のコンパイラ。

【請求項 8】

前記配分変換は、複数の前記配列について、ループの繰り返し処理を 1 回を行うのに進むアドレスであるストライドが異なるときに、前記ストライド比に応じて配分する

ことを特徴とする請求項 6 記載のコンパイラ。

【請求項 9】

前記配分変換は、内側ループを変換する際に、配分された各ループに対応する条件文を生成して、配分された各ループを同一の内側ループで実行されるように配分変換を行う

ことを特徴とする請求項 6 ~ 8 のいずれか 1 項に記載のコンパイラ。

【請求項 10】

前記配分変換は、内側ループを変換する際に、配分された各ループをそれぞれ生成するよう配分変換を行う

ことを特徴とする請求項 6～8 のいずれか 1 項に記載のコンパイラ。

【請求項 11】

前記ループ構造変換手段は、前記 x 回実行されるループを前記ループ回数が y 回のループに分割しようとしたとき、 x / y に余り z が出る場合は、余り z 回分をピーリング処理を行って前記二重ループ変換を行う

ことを特徴とする請求項 4 または 5 に記載のコンパイラ。

【請求項 12】

前記ループ構造変換手段は、

前記余り z が出る場合は、前記内側ループを y 回行うか z 回行うかを判定する条件文を生成し、前記二重ループ変換を行う

ことを特徴とする請求項 11 記載のコンパイラ。

【請求項 13】

前記最適化指定情報付加手段は、

当該ループが少なくとも何回実行されるかという最低実行回数情報を受け、

前記ループ構造変換手段は、

ループの実行回数が不定な場合は、前記最低実行回数情報に基づいて、最低実行回数分だけループを切り出し、切り出したループに対して前記二重ループ変換を行う

ことを特徴とする請求項 4 または 5 に記載のコンパイラ。

【請求項 14】

前記ループ構造変換手段は、ループの実行回数が不定な場合は、前記ループの実行回数の条件を判定して、その条件に応じてループの処理を行うようなコード生成を行う

ことを特徴とする請求項 4 または 5 に記載のコンパイラ

【請求項 15】

前記ループ構造変換手段は、ループ構造変換が不要と判定した場合は変換を行わないことを特徴とする請求項 4 または 5 記載のコンパイラ

【請求項 16】

前記最適化指定情報付加手段は、配列がキャッシュのラインサイズにアラインされているという情報を受け、

前記命令最適配置手段は、前記ループ処理に対して、前記ループ処理の繰り返し処理の一つ先のラインに記憶されるデータをプリフェッチするプリフェッチ命令を配置する

ことを特徴とする請求項 4 または 5 に記載のコンパイラ

【請求項 17】

前記最適化指定情報付加手段は、配列がラインのどの相対位置からアクセスを開始するかという情報を受け、

前記ループ構造変換手段は、前記情報に基づいて変換を行う

ことを特徴とする請求項 4 または 5 に記載のコンパイラ。

【請求項 18】

前記命令最適配置手段は、前記配列がキャッシュのラインサイズにアラインされていない場合には、前記ループ処理に対して、当該繰り返しの二つ先のラインに記憶されるデータをプリフェッチするプリフェッチ命令を配置する

ことを特徴とする請求項 4 または 5 記載のコンパイラ。

【請求項 19】

前記ループ構造変換手段は、前記配列がキャッシュのラインサイズにアラインされていない場合には、配列がラインのどの相対位置からアクセスを開始するかを判定し、その条件に応じてループの処理を行うようなコード生成を行う

ことを特徴とする請求項 4 または 5 に記載のコンパイラ。

【請求項 20】

前記ループ構造変換手段は、最内ループを 1 つかたまりとみなして、外側のループに

対して変換を行う
ことを特徴とする請求項4または5に記載のコンパイラ。

【請求項21】

前記最適化指定情報付加手段は、着目する配列に関する情報を受け、当該配列にのみ着
目し変換を行う
ことを特徴とする請求項4に記載のコンパイラ。

【書類名】明細書

【発明の名称】コンパイラ

【技術分野】

【0001】

本発明はコンパイラに関し、特にインターロックを起こす可能性のある命令を生成するコンパイラに関する。

【背景技術】

【0002】

近年、プロセッサの処理速度は急激に向かっているが、それに比べてメインメモリのアクセス速度向上は小さく、両者の速度差は年々大きくなっている。このため、情報処理装置の高速処理においてメモリアクセスがボトルネックとなることが従来指摘されている。

この問題を解消するために、記憶階層の考え方からキャッシュ機構が用いられている。キャッシュ機構では、プロセッサで必要とされるデータを主記憶から高速なキャッシュへ予め転送（プリフェッチ）しておく。これにより、プロセッサからのメモリアクセスに高速に対応することが可能である。

【0003】

しかし、プロセッサがキャッシュ上に無いデータにアクセスした場合にはキャッシュミスが発生してしまう。このため、主記憶からキャッシュへのデータの転送時間がかかるってしまうという問題がある。

ユーザがキャッシュを意識することなくプログラミングを行い、そのプログラムが実行されれば、このようなキャッシュミスが頻発することが想定される。その結果、キャッシュミスによるペナルティがプロセッサの性能を大きく劣化させることになる。そのため、コンパイラがキャッシュを考慮した最適化を行う必要がある。

【0004】

キャッシュ最適化の技術の一つとしてプリフェッチ命令の挿入が挙げられる（例えば、非特許文献1参照。）。プリフェッチ命令とは、あるメモリアドレスの参照が起こる前に、そのアドレスのデータを前もって主記憶からキャッシュへ転送しておくものである。プリフェッチ命令の挿入による最適化では、当該メモリアドレスの参照が起こる少し前のサイクルにプリフェッチ命令の挿入をおこなうものである。

【0005】

例えば、図31(a)に示すようなループ処理に対しては、図31(b)に示すようにデータが参照されるまでのレイテンシを考慮し、数イタレーション先で参照されるデータをプリフェッチするようにプリフェッチ命令（`d p r e f ()`）がループ内に挿入される。なお、ここでは、`i n t`型の配列aの要素は4バイト、キャッシュのラインサイズを128バイトとする。

【非特許文献1】Preeti Ranjan Panda, Hiroshi Nakamura, Nikil D. Dutt and Ale sandru Nicolau, "Augmenting Loop Tiling with Data Alignment for Improved Cache Performance", IEEE TRANSACTIONS ON COMPUTERS, FEBRUARY 1998, VOL. 48, NO. 2, p. 142-149

【発明の開示】

【発明が解決しようとする課題】

【0006】

しかし、図31(b)に示すコードでは、1イタレーションにつき配列aの参照とプリフェッチとがそれぞれ行われているが、参照は4byteずつしか行われないのでに対して、プリフェッチは1ライン（128byte）単位で行われる。

よって、1回のプリフェッチで32回分の参照に対応できるため、残り31回は無駄にプリフェッチが行われることになる。すなわち、同じラインのプリフェッチ命令を連續発行してしまっている。

【0007】

また、プロセッサによっては、`d p r e f`命令によるデータ転送中に、次の`d p r e f`

命令を実行しようとすると、前の `d pref` 命令による主記憶からキャッシュへのデータ転送が終了していないにもかかわらず、次の `d pref` 命令が発行されてしまい、インターロックが起こってしまう。

よって、上記のようにループの1イタレーションが短く、2つの `d pref` 命令の間隔が短いと、`d pref` 命令による主記憶からキャッシュへのデータ転送にかかる時間（レイテンシ）が顕在化し、かえって性能を悪化させてしまう。

【0008】

本発明は、上述の課題を解決するためになされたもので、インターロックを引き起こす可能性のある命令を無駄に発行せずに、プログラム実行時の処理速度を向上させるコンパイラを提供することを目的とする。

また、プログラム実行時にインターロックを引き起こさないコンパイラを提供することを目的とする。

【課題を解決するための手段】

【0009】

上記目的を達成するために、本発明に係るコンパイラは、実行時にインターロックを起こす可能性のある命令を備えたプロセッサ向けのコンパイラであって、入力プログラムに対し、ループ回数が x 回のループをループ回数が y 回のループに分割し、前記ループ回数が y 回のループを内側ループとし、ループ回数が x/y 回のループを外側ループとする二重ループ変換を行うループ構造変換手段と、前記二重ループ変換後のプログラムに対して、インターロックを起こす可能性のある命令の配置を行う命令最適配置手段としてコンピュータを機能させることを特徴とする。

【0010】

これにより、例えば図1に示すように、図31(a)に示すようなループ処理を二重ループ化し、最内ループの外側にプリフェッチ命令を挿入することができる。これにより、無駄なプリフェッチ実行がなくなる。よって、処理速度が向上する。また、ある `d pref` 命令が実行されてから次の `d pref` 命令が実行されるまでの間に、主記憶からキャッシュへのデータ転送にかかるレイテンシを隠蔽することが可能になり、インターロックが生じにくくなる。

【0011】

すなわち、本発明によると、ループを二重化することにより、内側ループの外側でインターロックを起こす可能性のある命令を実行するようすれば、当該命令を無駄に発行せずに、プログラム実行時の処理速度を向上させることができる。

【0012】

また、ループを二重化することにより、インターロックを起こす可能性のある命令を発行してから次のインターロックを起こす可能性のある命令までの間のサイクル数を確保することができる。このため、プログラム実行時にインターロックを引き起こしにくくなる。

【0013】

なお、本発明は、このような特徴的な命令を生成するコンパイラとして実現することができるだけでなく、コンパイラがコンピュータを機能させるための特徴的な手段を備えるコンパイル装置として実現したり、当該手段をステップとするコンパイル方法として実現したりすることもできる。そして、このようなコンパイラは、CD-ROM等の記録媒体やインターネット等の伝送媒体を介して流通させることができるのは言うまでもない。

【発明の効果】

【0014】

本発明によると、プログラム実行時の処理速度を向上させることができる。

また、プログラム実行時にインターロックを引き起こしにくくなる。

【発明を実施するための最良の形態】

【0015】

[システム構成]

図2は、本実施の形態に係るコンパイラシステムの構成を示す図である。コンパイラシステム148は、C言語等の高級言語で記述されたソースプログラム141を機械語の実行プログラム145に変換するソフトウェアシステムであり、コンパイラ149と、アセンブラー150と、リンク151とを含む。

【0016】

コンパイラ149は、キャッシュを備えるコンピュータのCPU(Central Processing Unit)をターゲットプロセッサとし、ソースプログラム141をアセンブラ言語で記述されたアセンブラファイル143に変換するプログラムである。コンパイラ149は、ソースプログラム141をアセンブラファイル143に変換する際に、キャッシュのラインサイズやレイテンシサイクル等に関する情報であるキャッシュパラメータ142や、後述するプロファイルデータ147に基づいて、最適化処理を行い、アセンブラファイル143を出力する。

【0017】

アセンブラー150は、アセンブラ言語で記述されたアセンブラファイル143を機械語で記述されたオブジェクトファイル144に変換するプログラムである。リンク151は、複数のオブジェクトファイル144を結合し、実行プログラム145を生成するプログラムである。

実行プログラム145の開発ツールとして、シミュレータ152およびプロファイル153が用意されている。シミュレータ152は、実行プログラム145をシミュレートし、実行時の各種実行ログデータ146を出力するプログラムである。プロファイル153は実行ログデータ146を解析し、プログラムの実行順序等を解析したプロファイルデータ147を出力するプログラムである。

【0018】

[コンパイラの構成]

図3は、コンパイラの構成を示す図である。コンパイラ149は、構文解析部182と、最適化情報解析部183と、一般最適化部184と、命令スケジューリング部185と、ループ構造変換部186と、命令最適配置部187と、コード出力部188とを含む。各構成処理部は、プログラムとして実現される。

【0019】

構文解析部182は、ソースプログラム141を入力として受け、構文解析処理を行った後、中間言語のプログラムを出力する処理部である。

最適化情報解析部183は、キャッシュパラメータ142、プロファイルデータ147、コンパイルオプションおよびプラグマなどの中間言語の最適化処理に必要な情報を読み込み、解析する処理部である。一般最適化部184は、中間コードに一般的な最適化処理を施す処理部である。命令スケジューリング部185は、命令の並びを最適化し、命令スケジューリングを行う処理部である。コンパイルオプションおよびプラグマはいずれもコンパイラに対する指示である。

【0020】

ループ構造変換部186は、一重ループを二重ループに変換する処理部である。命令最適配置部187は、変換された二重ループ内にプリフェッч命令を配置する処理部である。コード出力部188は、最適化された中間言語をアセンブラ言語に変換してアセンブラファイル143を出力する処理部である。

【0021】

[処理の流れ]

次に、コンパイラ149の実行する処理の流れについて説明する。図4は、コンパイラ149が実行する処理のフローチャートである。

【0022】

構文解析部182は、ソースプログラム141の構文解析を行い、中間コードを生成する(S1)。最適化情報解析部183は、キャッシュパラメータ142、プロファイルデータ147、コンパイルオプションおよびプラグマなどを解析する(S2)。一般最適化

部184は、最適化情報解析部183における解析結果に従い、一般的な中間コードの最適化を行う（S3）。命令スケジューリング部185は、命令のスケジューリングを行う（S4）。ループ構造変換部186は、中間コードに含まれるループ構造に着目し、必要であれば一重ループ構造を二重ループ構造に変換する（S5）。命令最適配置部187は、ループ構造内で参照されるデータをプリフェッチする命令を中間コードに挿入する（S6）。コード出力部188は、中間コードをアセンブラコードに変換し、アセンブラファイル143として出力する（S7）。

【0023】

構文解析処理（S1）、最適化情報解析処理（S2）、一般的な最適化処理（S3）、命令スケジューリング処理（S4）およびアセンブラコード出力処理（S7）は、一般的な処理と同様であるため、その詳細な説明はここでは繰返さない。

以下、ループ構造変換処理（S5）およびプリフェッチ命令配置処理（S6）について詳細に説明する。

【0024】

図5は、ループ構造変換処理（図4のS6）の詳細を説明するための図である。ループ構造変換部186は、ループ回数が即値で与えられており算出可能であるか、それ以外の変数等で与えられており算出不可能であるかを判断する（S11）。すなわち、ループ回数が固定であるか不定であるかを判断する。

ループ回数が不定の場合には（S11でNO）、プラグマまたはコンパイルオプションにより最低のループ回数の指定があるかまたはプログラム実行時に動的にループ回数を判定し、ループ分割をする旨の指定があるかについて判断する（S12）。

【0025】

いずれかの指定がある場合（S12でYES）またはループ回数が固定値の場合には（S11でYES）、ループ内で参照されている配列の添え字が解析可能か否かについて調べる（S13）。すなわち、ループカウンタがある規則性を持って変化している場合には解析可能であると判断される。例えば、ループカウンタの値がイタレーション内で書換えられるような場合には、解析不可能であると判断される。

【0026】

添え字が解析可能である場合には（S13でYES）、ループ処理内で参照される各配列について1イタレーションで参照される要素のバイト数を求め、そのうち最小の値LBを導出する（S14）。

【0027】

次に、キャッシュのラインサイズCSを値LBで割った値が1よりも大きいか否か判断する（S15）。CS/LBの値が1よりも大きい場合には（S15でYES）、ループ処理の配列がアラインされているか否かを調べる（S16）。配列がアラインされているか否かの判断は、プラグマやコンパイルオプション等によりアラインされているとの指示があるか否かにより判断される。

【0028】

配列がアラインされていない場合には（S17でNO）、「LB * LC / IC」がCSよりも大きいか否かについて判断する（S16）。ここで、LCは、レイテンシのサイクル数を示し、ICは1イタレーションあたりのサイクル数を示す。「LC / IC」は、ループを複数の最内ループに分割した場合の各ループのループ回数を示しており、「LB * LC / IC」は、各ループでのアクセス容量を示している。

【0029】

「LB * LC / IC」がラインサイズCSよりも大きい場合には、（S16でYES）、分割後の各ループ処理では1ラインサイズ以上の要素の参照が行われる。このため、分割要因をサイクルとし、各ループ処理を二重ループ化した際の最内ループのループ回数DTを次式（1）に従い導出する（S18）。

$$DT = (LC - 1) / IC + 1 \quad \dots (1)$$

「LB * LC / IC」がラインサイズCS以下の場合（S16でNO）または配列がア

ラインされている場合には（S17でYES）、分割要因をサイズとし、各ループ処理を二重ループ化した際の最内ループのループ回数DTを次式（2）に従い導出する（S19）。

【0030】

$$DT = (CS - 1) / LB + 1 \quad \dots (2)$$

最内ループのループ回数DTが導出処理（S18またはS19）後、最内ループのループ回数DTが1よりも大きいか否か判断される（S20）。DTが1の場合には（S20でNO）、最内ループのループ回数DTが1回であるため、ループを二重ループに構造変換する必要がない。このため、ループ構造変換処理（S5）を終了させる。

【0031】

最内ループのループ回数DTが2以上の場合には（S20でYES）、ループを二重ループに構造変換した場合の外側のループ構造が作成される（S21）。外側ループ構造を生成する際に、ピーリング処理が必要か否かを判断する（S22）。ピーリング処理およびピーリング処理が必要か否かの判断方法については後述する。

ピーリング処理が必要な場合には（S22でNO）、ピーリング処理を行い、ピーリングコードを生成する（S24）。その後、コンパイルオプション「-O」または「-Os」による指定があるか否かを調べる（S25）。ここで、コンパイルオプション「-O」は、プログラムサイズおよび実行処理速度ともに平均的なアセンブラーコードをコンパイラに出力させるための指示である。コンパイルオプション「-Os」は、プログラムサイズ抑制を重視したアセンブラーコードをコンパイラに出力させるための指示である。

【0032】

ピーリング処理する必要がないか（S22でYES）またはコンパイルオプション「-O」または「-Os」の指定がない場合には（S25でNO）、内側ループ（最内ループ）のループ回数の条件式を生成する（S23）。

コンパイルオプション「-O」または「-Os」の指定がある場合には（S25でYES）、ピーリングされたループ処理を二重ループに畳み込み、最内ループのループ回数の条件式を生成する（S26）。

【0033】

最内ループのループ回数条件生成処理（S23、S26）の後、最内ループにおける参照の対象配列は1つであるか否か調べられる（S27）。最内ループにおける参照の対象配列が1つの場合には、（S27でYES）、ループ構造変換処理（S5）を終了する。

最内ループにおける参照の対象配列が2つ以上ある場合には（S27でNO）、最内ループの分割個数を導出し、分割後の各最内ループのループ回数の比率を決定する（S28）。その後、分割後の最内ループ回数DTを分割個数で割った値が1よりも大きいか否か判断する（S29）。すなわち、当該値が1以下の場合には（S29でNO）、分割後の各ループ回数が1回以下であるため、分割する意味がない。このため、ループ構造変換処理（S5）を終了させる。

【0034】

当該値が1よりも大きい場合には（S29でYES）、分割後の各ループ回数が2回以上である。この場合には、コンパイルオプション「-O」または「-Ot」による指定があるか否かを調べる（S30）。コンパイルオプション「-Ot」は、実行処理速度向上を重視したアセンブラーコードをコンパイラに出力させるための指示である。

コンパイルオプション「-O」または「-Os」による指定がある場合には（S30でYES）、後述する実行処理速度向上を重視したコピー型内側ループ分割処理（S31）を実行し、ループ構造変換処理（S5）を終了する。

【0035】

コンパイルオプション「-O」または「-Os」による指定がない場合には（S30でNO）、後述するプログラムサイズ抑制を重視した条件型内側ループ分割処理（S32）を実行し、ループ構造変換処理（S5）を終了する。

【0036】

図6は、コピー型内側ループ分割処理（図5のS31）の詳細を示すフローチャートである。

最内ループのループ回数DTを分割個数で割った値を細分割後内側ループ回数とする（S41）。次に、内側ループを分割個数分だけ複製し、生成する（S42）。その後、細分割後の各内側ループ回数を細分割後内側ループ回数に修正する（S43）。さらに、DTを分割個数で割った剩余を細分割後の先頭ループのループ回数に加算し（S44）、コピー型内側ループ分割処理を終了する。

【0037】

図7は、条件型内側ループ分割処理（図5のS32）の詳細を示すフローチャートである。

最内ループのループ回数DTを分割個数で割った値を細分割後内側ループ回数とする（S51）。次に、内側ループ回数条件の切換えswitchテーブルを生成する（S52）。すなわち、内側ループ回数を順次切り替えるように、C言語で言うところのswitch文を生成する。なお、if文であってもよい。

【0038】

テーブル生成後、細分割後の各内側ループ回数条件を細分割後の内側ループ回数に修正する（S53）。その後、DTを分割個数で割った剩余を細分割後の先頭ループの回数条件に加算し（S54）、条件型内側ループ分割処理を終了する。

【0039】

図8は、プリフェッチ命令配置処理（図4のS6）の詳細を示すフローチャートである。

- プリフェッチ命令配置処理では、すべてのループについて以下の処理を繰返す（ループA）。まず、着目しているループが命令挿入対象のループであるか否かを調べる（S61）。命令挿入対象のループであるか否かの情報は、ループ構造変換部186の解析結果より取得される。

命令挿入対象のループの場合には（S61でYES）、そのループに対して条件型ループ分割が行われているか否かを調べる（S62）。条件型ループ分割が行われていれば、各条件文における命令挿入位置を解析し（S63）、プリフェッチ命令を挿入する（S64）。命令挿入対象のループに対して条件型ループ分割が行われていなければ（S62でNO）、そのループに対してコピー型ループ分割が行われているか否かを調べる（S65）。コピー型ループ分割が行われていれば（S65でYES）、そのループの手前の命令挿入位置を解析する（S66）。その後、プリフェッチ命令が挿入される（S67）。ピーリングされたループの場合には（S68でYES）、当該ループの手前に命令挿入するように命令挿入位置が解析され（S69）、その位置にプリフェッチ命令が挿入される（S70）。

【0040】

図9は、プリフェッチ命令挿入処理（図8のS64、S67およびS70）の詳細を示すフローチャートである。

命令挿入処理では、挿入命令、挿入位置、挿入アドレス等からなる情報リストがすべて空になるまで以下を繰返す（ループB）。

プリフェッチ命令を挿入しようとしている配列要素がアライン済みであるか否かを判断する（S72）。アラインされていなければ（S72でNO）、サイクル要因に従ってループ分割されたものであるのか、サイズ要因に従ってループ分割されたものであるのかを調べる（S73）。

【0041】

アライン済みであるか（S72でYES）またはサイクル要因でループ分割されたものであれば（S73でYES）、1ライン先のデータをプリフェッチする命令を挿入する（S74）。アラインされておらず、かつサイズ要因でループ分割されたものであれば（S73でNO）、2ライン先のデータをプリフェッチする命令を挿入する（S75）。最後に、解析済みの情報を情報リストから削除する（S76）。

[コンパイルオプション]

コンパイラシステム148では、コンパイラに対するコンパイルオプションとして、オプション「-fno-loop-tilling-dpref」が用意される。このオプションが指定されれば、プラグマの指定に関わらず、ループに対する構造変換は行わない。本オプションの指定がなければ、構造変換の実施はプラグマ指定の有無に従う。

【0042】

[プラグマ指定]

本指定は、直後のループに対するものである。

【0043】

プラグマ「#pragma _loop_tiling_dpref 変数名 [, 変数名]」により変数が指定された場合には、プラグマ指定された変数のみに着目してループ分割を行う。指定する変数は、配列でも、ポインタでもよい。

プラグマ「#pragma _loop_tiling_dpref_all」によりループが指定された場合には、ループ内で参照される配列の全てに着目して構造変換が行われる。

【0044】

以下、いくつかの具体的曲面におけるループ分割処理について説明する。なお、以降の処理では、説明の簡単化のためC言語によるプログラム記述を行っているが、実際には中間言語による最適化処理が行われる。

【0045】

[シンプルループ分割]

図10は、ピーリングが必要ない場合のシンプルループ分割処理について説明するための図である。

【0046】

図10 (a) に示すようなソースプログラム282が入力された場合について考える。このソースプログラム282では、配列Aの要素が順次参照され、変数sumに加算される。ここで、配列Aの各要素のサイズは4バイトであるものとし、キャッシュの1ラインサイズは128バイト（以降の説明でも、キャッシュのラインサイズは128バイトであるものとする。）であるものとする。すなわち、キャッシュの1ラインには配列Aの要素が32個記憶される。また、ソースプログラム282に含まれるループのイテレーションの回数128回は、32の整数倍である。このため、ソースプログラム282は、図10 (b) のプログラム284に示すように、二重ループに構造変換することができる。すなわち、最内ループでは32回の繰り返し処理を行い、その外のループでは、最内ループを4回繰返すループ処理を行う。最内ループ処理ではキャッシュの1ライン分のデータが参照される。その後、図10 (c) のプログラム286に示されるように、最内ループの実行前に、プリフェッч命令(pref (&A[i+32]))が挿入される。プリフェッч命令を挿入することにより、最内ループ実行時には、当該ループで参照される配列Aの要素がキャッシュに乗っていることになる。

【0047】

図11～図14は、ピーリングが必要ないシンプルループ分割処理における中間言語の推移を説明するための図である。

図11は、図10 (a) と同様に、ピーリングが必要ない場合のソースプログラムの一例を示す図である。図12は、図11に示したソースプログラム240に対応する中間言語のプログラムである。[BGNBLK]と[ENDBLK]とで挟まれた内部の命令列が1つの基本ブロックに対応しており、[BGNBLK] B1で始まる基本ブロックがforループの直前までの処理を示しており、[BGNBLK] B2で始まる基本ブロックがforループを示しており、[BGNBLK] B3で始まる基本ブロックがforループの後の処理を示している。

【0048】

図13は、図12に示された中間言語のプログラムを二重ループに構造変換した後の

間言語のプログラムを示している。[BGNBBLK] B2で始まる基本ブロックが最内ループに対応しており、[BGNBBLK] B4および[BGNBBLK] B5で始まるループがその外側のループに対応している。

【0049】

図14は、図13に示された中間言語のプログラムにプリフェッチ命令を挿入した後の中間言語のプログラムを示している。プログラム270では、[BGNBBLK] B4で始まる基本ブロックの内部にプリフェッチ命令(pref)が新たに挿入されている。

【0050】

図15は、ピーリングが必要な場合のシンプルループ分割処理について説明するための図である。

図15(a)に示すようなソースプログラム292が入力された場合について考える。このソースプログラム292では、配列Aの要素が順次参照され、変数sumに加算される。ここで、配列Aの各要素のサイズは4バイトであるものとする。すなわち、キャッシュの1ラインには配列Aの要素が32個記憶される。また、ソースプログラム292に含まれるループのイテレーションの回数は140回であるものとする。すなわち、1ラインに記憶される配列Aの要素数32で割った場合に余りが出る数である。

【0051】

このような場合には、図15(b)に示すプログラム294のように、140を32で割った余りのループ回数をピーリングし、それ以外の部分を図10(b)と同様に二重ループ構造に構造変換する。その後、ピーリングされた部分を二重ループ構造に含ませるためのピーリング畳み込み処理を行い、図15(c)に示すようなプログラム296が得られる。すなわち、通常状態では最内ループで32回の繰り返し処理が行われ、最後に最内ループが実行される場合には、残りの12(=140-128)回の繰り返し処理が行われる。その後、図15(d)のプログラム298に示されるように、最内ループの実行前に、プリフェッチ命令(pref(&A[i+32]))が挿入される。

【0052】

[複数配列アクセスが存在する場合(ピーリング必要なし)]

図16は、ループ内に複数の配列アクセスが存在する場合のループ分割処理について説明するための図である。

【0053】

図16(a)に示すようなソースプログラム301が入力された場合について考える。このソースプログラム301では、配列Aおよび配列Bの要素が順次参照され、当該要素同士の積が変数sumに加算される。ここで、配列Aおよび配列Bの各要素はそれぞれ4バイトであるものとする。すなわち、キャッシュの1ラインには配列Aの要素が32個記憶される。または、配列Bの要素が32個記憶される。すなわち、1ラインに格納される要素数は配列Aと配列Bとで同じである。また、ソースプログラム301に含まれるループのイテレーションの回数128回は、32の整数倍である。このため、ソースプログラム301は、図16(b)のプログラム302に示すように、ピーリングをすることなく二重ループに構造変換することができる。

【0054】

複数配列アクセスが存在する場合の二重ループ構造は、コピー型と呼ばれる実行処理速度を向上させるための最適化と、条件型と呼ばれるプログラムサイズを小さくするための最適化との二種類がある。

まず、コピー型の最適化について説明する。プログラム302に含まれる最内ループのループ回数を配列Aと配列Bとの要素の大きさの比で分割する。ここでは、配列Aと配列Bとはともに同じ要素の大きさである。したがって、図16(c)に示すプログラム303のように最内ループを二等分し、ループ回数が16回の最内ループ2つに分割する。次に、図16(d)のプログラム304に示すように、各最内ループの直前にプリフェッチ命令を挿入する。最初の最内ループの直前には、1ライン分の配列Aの要素をプリフェッチするためのプリフェッチ命令(pref(&A[i+32]))が挿入され、2番目

の最内ループの直前には、1ライン分の配列Bの要素をプリフェッチするためのプリフェッチ命令（`d pref (&B [i + 32])`）が挿入される。

【0055】

このようにプリフェッチ命令間にループ処理を挿入させることにより、異なる配列に対するプリフェッチ命令が連続することが無くなり、プリフェッチ命令実行によるレイテンシを隠蔽することができる。このため、実行処理速度を向上させることができる。

次に、条件型の最適化について説明する。条件型の場合も、コピー型の場合と同様にして最内ループのループ回数を配列Aと配列Bとの要素の大きさの比で分割する。ただし、プログラム303のように最内ループを2つ並べるのではなく、図16(e)に示すプログラム305のように最内ループの個数は1つであり、そのループ回数を条件分岐させようとしている。すなわち、変数K=1の場合と、K=0の場合とで最内ループのループ回数Nを変えるようにしている。ただし、この例では変数Kの値に関係なく最内ループの回数Nは16回となっている。次に、図16(f)に示すプログラム306のように、K=1の場合には配列Aの要素を1ライン分プリフェッチし、K=0の場合には配列Bの要素を1ライン分プリフェッチするように条件分岐式およびプリフェッチ命令の挿入が行われる。なお、ここでは、最適化によりループ回数Nは即値16に置き換えられている。

【0056】

このように、最内ループの個数を1つにし、条件分岐式で最内ループのループ回数およびプリフェッチ命令を変えるようにすることにより、最終的に生成される機械語命令のプログラムサイズを小さくすることができる。ただし、条件分岐処理があるため、コピー型に比べて処理速度が多少遅くなる可能性がある。

【0057】

[複数配列アクセスが存在する場合（ピーリング必要）]

図17は、ループ内に複数の配列アクセスが存在する場合のループ分割処理について説明するための図である。

【0058】

図17(a)に示すようなソースプログラム311が入力された場合について考える。このソースプログラム311では、配列Aおよび配列Bの要素が順次参照され、当該要素同士の積が変数sumに加算される。ここで、配列Aおよび配列Bの各要素はそれぞれ4バイトであるものとする。すなわち、キャッシュの1ラインには配列Aの要素が32個記憶される。または、配列Bの要素が32個記憶される。すなわち、1ラインに格納される要素数は配列Aと配列Bとで同じである。また、ソースプログラム311に含まれるループのイタレーションの回数は140回であるものとする。

【0059】

したがって、ソースプログラム311を二重ループに構造変換する場合には、図15(b)に示したプログラム294と同様、図17(b)に示すようにピーリング処理されたプログラム312が生成される。

コピー型の最適化を行う際には、配列Aと配列Bとの要素の大きさの比で最内ループを分割する。すると、図17(c)に示すプログラム313が生成される。次に、図17(d)のプログラム314に示すように、最初の最内ループの直前には、1ライン分の配列Aの要素をプリフェッチするためのプリフェッチ命令（`d pref (&A [i + 32])`）が挿入され、2番目の最内ループの直前には、1ライン分の配列Bの要素をプリフェッチするためのプリフェッチ命令（`d pref (&B [i + 32])`）が挿入される。なお、ピーリング処理された最終ループの直前にはプリフェッチ命令は挿入されない。これは、その前の二重ループ処理におけるプリフェッチ命令実行により所望のデータがキャッシュにプリフェッチされているからである。

【0060】

条件型の最適化を行う際には、プログラム312に対してピーリング畳み込み処理を行い、図17(e)に示されるようなプログラム315を得る。ピーリング畳み込み処理は、図15を参照して説明したものと同様である。次に、最内ループのループ回数を配列A

と配列Bとの要素の大きさの比で分割し、当該ループ回数を条件分岐させるように図17(f)に示すプログラム316を作成する。プログラム316においては、変数Kの値を交互に変更させ、変数Kの値に対応するようにループカウンタNの値を変化させる。次に、図17(g)のプログラム317に示すように、Kの値の変化に伴い、配列Aおよび配列Bの要素を1ライン分ずつ交互にプリフェッチするように、条件分岐式中にプリフェッチ命令を挿入する。

【0061】

このように、ピーリングが必要な場合であっても、コピー型の場合にはピーリングの部分を二重ループとは別のループにし、条件型の場合には、条件分岐式によりピーリングの場合のループカウンタの回数を変えるようにすることにより、ループ内に複数の配列アクセスがあり、かつピーリングが必要な場合であっても、プリフェッチによるレイテンシを考慮した最適化を行なうことができる。

【0062】

[サイズが異なる複数配列アクセスが存在する場合(ピーリング必要なし)]

図18は、ループ内に複数の配列アクセスが存在し、かつ配列の要素のサイズがすべて同じではない場合のループ分割処理について説明するための図である。

【0063】

図18(a)に示すようなソースプログラム321が入力された場合を考える。ここで、配列Aの要素は4バイト、配列Bの要素は2バイトとする。すなわち、キャッシュの1ラインには配列Aの要素が32個、配列Bの要素が64個記憶される。

この場合、要素サイズの小さい配列Bに着目し、配列Bの要素に応じたループの構造変換を行う。すなわち、図18(b)のプログラム322のように、最内ループのループ回数を1ラインに収まるキャッシュBの要素数64にし、二重ループに構造変換する。最内ループでは、配列Bに関しては1ライン分の要素が消費されるが、配列Aに関しては2ライン分の要素が消費されることになる。このため、最内ループ処理を実行するためには合計3ライン分のデータが必要になる。

【0064】

このため、コピー型の最適化を行う際には、図18(c)のプログラム323に示すように、最内ループを3つに分割し、図18(d)のプログラム324に示すように、各最内ループの直前にプリフェッチ命令を挿入する。ここでは、1番目の最内ループの直前には、2ライン先の配列Aの要素をプリフェッチするプリフェッチ命令(dpref(&A[i+64]))を挿入し、2番目の最内ループの直前には3ライン先の配列Aの要素をプリフェッチするプリフェッチ命令(dpref(&A[i+96]))を挿入し、3番目の最内ループの直前には1ライン先の配列Bの要素をプリフェッチするプリフェッチ命令(dpref(&B[i+64]))を挿入している。また、3つの最内ループのループ回数を処理順に22、21および21としている。これは、最外ループの条件分岐判断が3番目の最内ループ実行後に行われるため、3番目の最内ループのループ回数を少なくすることにより、全体としての処理速度を向上させるためである。

【0065】

また、条件型の最適化を行う際には、図18(e)のプログラム325に示すように、1回の最内ループ処理につき、変数Kの値を0から2までの範囲内で更新させ、変数Kの値による条件分岐処理により最内ループのループ回数Nを22、21および21のうちのいずれかに設定する。その後、ループ回数Nの最内ループを実行させる。次に、図18(f)のプログラム326に示すように、変数Kの値が0の場合にはプリフェッチ命令(dpref(&A[i+64]))を実行させ、変数Kの値が1の場合にはプリフェッチ命令(dpref(&A[i+96]))を実行させ、変数Kの値が2の場合にはプリフェッチ命令(dpref(&B[i+64]))を実行させるように最適化を行う。

[サイズが異なる複数配列アクセスが存在する場合(ピーリング必要)]

図19は、ループ内に複数の配列アクセスが存在し、かつ配列の要素のサイズがすべて同じではない場合のループ分割処理について説明するための図である。

【0066】

図19 (a) に示すソースプログラム331は、図18 (a) に示したソースプログラム321と同様、配列Aの要素は4バイト、配列Bの要素は2バイトである。したがって、ソースプログラム321とループ回数が異なるのみである。ソースプログラム321のループを二重ループに構造変換し、ループ回数140を配列Bの1ライン分の要素数64で割った余りをピーリング処理すると、プログラム322が得られる。コピー型の最適化処理を行う場合には、図18 (c) および図18 (d) を参照して説明したように、二重ループの最内ループを3分割し、プリフェッチ命令を挿入することにより、図19 (c) に示すプログラム333が得られる。条件型の最適化処理を行う場合には、図18 (e) および図18 (f) を参照して説明したように、条件分岐式によりループ回数およびプリフェッチ命令を制御し、最終的に図19 (e) に示すプログラム335が得られる。

【0067】

[ストライドが異なる複数配列アクセスが存在する場合]

図20は、ループ内にストライドが異なる複数の配列アクセスが存在する場合のループ分割処理について説明するための図である。

【0068】

ストライドとは、ループ処理における配列要素の増分値（アクセス幅）のことを示す。図20 (a) に示すようなソースプログラム341が入力された場合を考える。ここで、配列Aの要素および配列Bの要素はともに4バイトであるものとする。ソースプログラム341では、ループのイタレーションごとに、配列Aの要素は1ずつ増加するのに対し、配列Bの要素は2ずつ増加する。すなわち、配列Bのアクセス幅は配列Aのアクセス幅の2倍である。最小アクセス幅の配列Aに着目すると、1ラインには配列Aの要素が32個収まる。このため、最内ループのループ回数を32回とした二重ループへの構造変換を行うと、図20 (b) に示すプログラム342が得られる。最内ループでは、配列Aに関しては1ライン分の要素が消費されるが、配列Bに関しては2ライン分の要素が消費されることになる。このため、最内ループ処理を実行するためには合計3ライン分のデータが必要になる。

【0069】

よって、コピー型の最適化を行う際には、図20 (c) のプログラム343に示すように、最内ループを3つに分割し、図20 (d) のプログラム344に示すように、各最内ループの直前にプリフェッチ命令を挿入する。ここでは、1番目の最内ループの直前には、1ライン先の配列Aの要素をプリフェッチするプリフェッチ命令 (d p r e f (& A [i + 32])) を挿入し、2番目の最内ループの直前には2ライン先の配列Bの要素をプリフェッチするプリフェッチ命令 (d p r e f (& B [i * 2 + 64])) を挿入し、3番目の最内ループの直前には3ライン先の配列Bの要素をプリフェッチするプリフェッチ命令 (d p r e f (& B [i * 2 + 96])) を挿入している。

【0070】

また、条件型の最適化を行う際には、図20 (e) のプログラム345に示すように、1回の最内ループ処理につき、変数Kの値を0から2までの範囲内で更新させ、変数Kの値による条件分岐処理により最内ループのループ回数Nを11、11および10のうちのいずれかに設定する。その後、ループ回数Nの最内ループを実行させる。次に、図20 (f) のプログラム346に示すように、変数Kの値が0の場合にはプリフェッチ命令 (d p r e f (& A [i + 32])) を実行させ、変数Kの値が1の場合にはプリフェッチ命令 (d p r e f (& B [i * 2 + 64])) を実行させ、変数Kの値が2の場合にはプリフェッチ命令 (d p r e f (& B [i * 2 + 96])) を実行させるように最適化を行う。

【0071】

[ループ回数が不定な場合]

図21は、ループ回数が不定なループ処理のループ分割処理を説明するための図である

【0072】

図21(a)に示すソースプログラム351が入力された場合を考える。ソースプログラム351に含まれるループ回数は変数Va1により特定され、コンパイル時には不定である。しかし、最低128回は繰り返し処理が行われることがプラグマ指定「#pragma min_iteration=128」により、保証されている。ここで、配列Aは4バイトであるものとする。すなわち、キャッシュの1ラインには配列Aの要素が32個記憶される。

【0073】

プラグマ指定に従い、ループ処理を最初の128回のループ処理と、それ以降の変数Va1で特定されるループ回数のループ処理とに分割し、それぞれをシンプルループの場合と同様に二重ループ化すると図21(b)に示すプログラム352が得られる。

コピー型の最適化処理を行う場合には、プログラム352の最内ループの直前に1ライン先の配列Aの要素をプリフェッчするためのプリフェッч命令(dpref(&A[i+32]))を挿入することにより、図21(c)に示すプログラム353が得られる。

【0074】

条件型の最適化処理を行う場合には、後半のループ処理をピーリング畳み込みし、最外ループ回数が128回になるまでは、最内ループの回数を32回にし、それ以降は最内ループの回数を(Va1-128)回に設定する分岐命令を挿入する。すると、図21(d)に示すようなプログラム354が得られる。

最後に、最内ループの実行前にプリフェッч命令(dpref(&A[i+32]))を挿入することにより図21(e)に示すようなプログラム355が得られる。

【0075】

図22は、ループ回数が不定なループ処理のループ分割処理を説明するための他の図である。

図22(a)に示すソースプログラム361が入力された場合を考える。ソースプログラム361に含まれるループ回数は変数Nにより特定され、コンパイル時には不定である。また、ソースプログラム361は、ソースプログラム351と異なり、最低のループ回数を示すプラグマ指定がない。

【0076】

ループ回数が小さなループ処理に対してループの構造変換を行い、最適化を行ったとしても、最適化の効果が表れにくい。このため、このような場合には、最適化の効果を高めるために、ループ回数があるしきい値よりも大きければ最適化されたループ処理を実行し、それ以外の場合には通常のループ処理を実行するようとする。例えば、あるしきい値を1024とした場合には、図22(b)のプログラム362に示されるように、ループ回数Nが1024を超える場合には、最初の1024回のループ処理については二重ループを実行し、残りの回数のループ処理については、ピーリングされたループ処理を行うようになる。また、ループ回数Nが1024以下の場合には、二重ループは実行せずに、ピーリングされたループ処理を実行するようになる。その後、二重ループの最内ループの直前にプリフェッч命令(dpref(&A[i+32]))を挿入することにより図22(c)に示すような最適化されたプログラム363が生成される。

【0077】**[ループ分割が不要な場合]**

図23は、ループ分割が不要な場合の最適化処理を説明するための図である。図23(a)に示すソースプログラム371が入力された場合には、ループ中で1ライン分のデータ(A[i]～A[i+31])を完全に使い切ってしまう。このような場合には、二重ループ化する必要はない。このため、図23(b)に示すプログラム372のようにループの先頭にループ内で使用されるデータの1ライン先のデータをプリフェッчするプリフェッч命令(dpref(&A[i+32]))を挿入することにより最適化が行われる。

【0078】

また、ループ内の処理サイクル数がプリフェッチ命令で必要とされる処理サイクル数よりも大きいような場合にも、ループを二重化する必要はなく、ループの先頭にプリフェッチ命令を挿入してもプリフェッチ命令のレイテンシは隠蔽することができる。

【0079】

[ループ内でアクセスされる要素がアラインされていない場合]

図24および図25は、ループ内でアクセスされる要素が主記憶上で適切にアラインされていない場合のループ分割処理を説明するための図である。これまでの説明では、ループ内でアクセスされる要素が主記憶上で適切にアラインされている場合を想定して話を進めてきた。アラインされていることが予めプログラマや、コンパイルオプションの指定によりわかっている場合には、上述の例で説明したような最適化が行われる。

【0080】

しかし、一般的にはコンパイラは、それらの要素がアラインされているか否かは実行時まではわからない。このため、コンパイラは、ループ内アクセス要素が主記憶上で適切にアラインされていないことを前提として最適化を行う必要がある。

すなわち、図24(a)に示すようなソースプログラム381が与えられた場合に、配列Aの要素サイズを4バイトとすると、図10を参照して説明したシンプルループ分割と同様にして、最適化が行われる。ただし、要素がアラインされていないことを前提としているため、最内ループの前に挿入されるプリフェッチ命令(dpref(&A[i+64]))は2ライン先の配列Aの要素をプリフェッチ指定している。また、ループ処理に先立って、ループ内でアクセスされる配列の要素A[0]～A[63]を確保するために、プリフェッチ命令(dpref(&A[0]))およびdpref(&A[32]))がプリフェッチのレイテンシを十分隠蔽できる位置に挿入され、図24(b)に示すようなプログラム382が生成される。

【0081】

また、図25(a)に示されるようなソースプログラム391が与えられた場合には、図15と同様にして、ピーリング処理された部分のループを畳み込んだ後に、2ライン先の配列Aの要素をプリフェッチする命令(dpref(&A[i+64]))が挿入される。また、プログラム382と同様にプリフェッチ命令(dpref(&A[0]))およびdpref(&A[32]))が挿入され、図25(b)に示すような最適化されたプログラム392が生成される。

【0082】

[動的アライン解析コードの挿入による構造変換分割]

図26は、アラインされていない配列要素を動的に特定して、ループ処理の最適化を行う処理について説明するための図である。図26(a)に示すソースプログラム401が入力された場合を考える。ここで、配列Aの要素は4バイトであるものとする。

【0083】

配列Aの先頭アドレス(要素A[0]のアドレス)の所定のビットがキャッシュのラインを示しており、そのビット内のさらにあるビットは、ラインの先頭からのオフセットを示している。したがって、「A&Mask」というビット同士の論理演算を行うことにより、ラインの先頭からのオフセットを取り出すことができる。ここで、マスク値Maskはあらかじめ定められた値である。配列Aの先頭アドレスから取り出されたオフセット値を予め定められた補正值Corだけ右シフトすることにより、配列Aの先頭要素A[0]が1ライン内で先頭から何番目に位置しているかがわかる。よって、次式(3)にしたがって、ライン上でアラインされていない要素の数nを求めることができる。

【0084】

$$n = 32 - (A \& Mask) >> Cor \quad \dots (3)$$

すなわち、図27に示すように、キャッシュ431にフェッチした場合に、アラインされない配列Aの要素(A[0]～A[n-1])とアラインされる配列Aの要素とが区別されることになる。

【0085】

したがって、図26（b）のプログラム402に示すように、式（3）に従いアラインされていない配列Aの要素数nを求める。次に、要素数nに従って、アラインされていない配列Aの要素（A[0]～A[n-1]）についてのループ処理を行う。その後、アラインされている配列Aの要素（A[n]以降の要素）については、図10に示したシンプルループ分割の場合と同様に二重ループ化を行う。

【0086】

その後、ピーリングされているループ405について、畳み込み処理を行うと、図26（c）に示すようなプログラム403が生成される。また、図26（d）に示すように、プリフェッヂ命令（`d pref (&A[i+32])`）を挿入することにより、最適化されたプログラム404が得られる。

【0087】

[プロファイル情報を用いた構造変換分割]

図28は、アラインされていない配列要素をプロファイル情報を用いて特定し、ループ処理の最適化を行う処理について説明するための図である。アラインされていない配列の要素数を図26のように計算から求めるのではなく、プロファイル情報から取得する。取得したアラインされていない配列の要素数Nに基づいて、図26に示したのと同様の処理を行い、図28（a）に示すソースプログラム411を図28（b）に示すプログラム412のように変換する。その後、ピーリングされたループ部分を畳み込み、図28（c）に示すプログラム413を得る。最後に、図28（d）に示すプリフェッヂ命令を挿入することにより最適化されたプログラム414を得る。

[最内ループ以外のループに対する構造変換]

図29は、最内ループ以外のループに対する構造変換について説明するための図である。

【0088】

図29（a）に示すソースプログラム421が与えられた場合を考える。ソースプログラム421では、二重ループ処理が行われており、最内ループ処理424で参照される配列Aの要素は1バイトであるものとする。最内ループ処理424のループ回数は4回であるため、最内ループ処理424では配列Aの要素が4バイト分参照される。したがって、最内ループ処理424で参照される要素のバイト数が小さいため、このような場合には、最内ループ処理424を1つの固まりとして考え、最外ループを、図29（b）に示すプログラム422のように、二重ループに構造変換する。その後、2番目のループ処理の実行前にキャッシュの1ライン分の配列Aの要素をプリフェッヂする命令（`d pref (&A[j+128])`）が挿入され、図29（c）に示すような最適化されたプログラム423が得られる。

【0089】

[プラグマ「#pragma _loop_tiling_pref 変数名 [, 変数名]」による変数指定]

図30は、プラグマ「#pragma _loop_tiling_pref 変数名 [, 変数名]」により変数が指定された場合の最適化処理について説明するための図である。図30（a）に示すように、プラグマ「#pragma _loop_tiling_pref b」との指定がソースプログラム中に含まれる場合には、ループ内の配列bのみに着目して構造変換が行われ、配列aは無視される。従って、図30（b）に示すような二重ループ化が実行され、配列bをプリフェッヂする命令のみが挿入される。

【0090】

以上説明したように、本実施の形態に係るコンパイラシステムによると、ループ処理を二重化し、最内ループの外側でプリフェッヂ命令を実行するようにしている。このため、無駄なプリフェッヂ命令の発行を防ぐことができ、プログラム実行時の処理速度を向上させることができる。また、ループ処理を二重化することにより、プリフェッヂ命令を実行してから次のプリフェッヂ命令を実行するまでのサイクル数を確保することができる。このため、レイテンシを隠蔽し、インターロックを防ぐことができる。

【0091】

以上、本発明の実施の形態に係るコンパイルシステムについて、実施の形態に基づいて説明したが、本発明は、この実施の形態に限定されるものではない。

例えば、命令最適配置部187で配置される命令は、プリフェッチ命令に限られず、インターロックを起こさないようにする命令であれば、他の命令であってもよい。

また、キャッシュを備えないコンピュータのCPUをターゲットプロセッサとして、各種処理のレイテンシを隠蔽し、インターロックを防ぐようなコードを出力するコンパイルシステムであってもよい。

【産業上の利用可能性】

【0092】

本発明はインターロックを起こす可能性のある命令の発行を制御するコンパイラー、OS(Operating System)、プロセッサで実行されるプロセス等に適用できる。

【図面の簡単な説明】

【0093】

【図1】本発明によるループ処理の構造変換を説明するための図である。

【図2】本実施の形態に係るコンパイラシステムの構成を示す図である。

【図3】コンパイラの構成を示す図である。

【図4】コンパイラが実行する処理のフローチャートである。

【図5】ループ構造変換処理の詳細を説明するための図である。

【図6】コピー型内側ループ分割処理の詳細を示すフローチャートである。

【図7】条件型内側ループ分割処理の詳細を示すフローチャートである。

【図8】プリフェッチ命令配置処理の詳細を示すフローチャートである。

【図9】プリフェッチ命令挿入処理の詳細を示すフローチャートである。

【図10】ピーリングが必要ない場合のシンプルループ分割処理について説明するための図である。

【図11】ピーリングが必要のない場合のソースプログラムの一例を示す図である。

【図12】図11に示したソースプログラムに対応する中間言語のプログラムを示す図である。

【図13】図12に示された中間言語のプログラムを二重ループに構造変換した後の中間言語のプログラムを示す図である。

【図14】図13に示された中間言語のプログラムにプリフェッチ命令を挿入した後の中間言語のプログラムを示す図である。

【図15】ピーリングが必要な場合のシンプルループ分割処理について説明するための図である。

【図16】ループ内に複数の配列アクセスが存在する場合のループ分割処理について説明するための図である。

【図17】ループ内に複数の配列アクセスが存在する場合のループ分割処理について説明するための図である。

【図18】ループ内に複数の配列アクセスが存在し、かつ配列の要素のサイズがすべて同じではない場合のループ分割処理について説明するための図である。

【図19】ループ内に複数の配列アクセスが存在し、かつ配列の要素のサイズがすべて同じではない場合のループ分割処理について説明するための図である。

【図20】ループ内にストライドが異なる複数の配列アクセスが存在する場合のループ分割処理について説明するための図である。

【図21】ループ回数が不定なループ処理のループ分割処理を説明するための図である。

【図22】ループ回数が不定なループ処理のループ分割処理を説明するための他の図である。

【図23】ループ分割が不要な場合の最適化処理を説明するための図である。

【図24】ループ内でアクセスされる要素が主記憶上で適切にアラインされていない場合のループ分割処理を説明するための図である。

【図25】ループ内でアクセスされる要素が主記憶上で適切にアラインされていない場合のループ分割処理を説明するための図である。

【図26】アラインされていない配列要素を動的に特定して、ループ処理の最適化を行う処理について説明するための図である。

【図27】アラインされていない配列要素を説明するための図である。

【図28】アラインされていない配列要素をプロファイル情報を用いて特定し、ループ処理の最適化を行う処理について説明するための図である。

【図29】最内ループ以外のループに対する構造変換について説明するための図である。

【図30】プラグマ「#pragma __loop_tiling_dpref 変数名 [, 変数名]」により変数が指定された場合の最適化処理について説明するための図である。

【図31】従来の最適化技術の問題点を説明するための図である。

【符号の説明】

【0094】

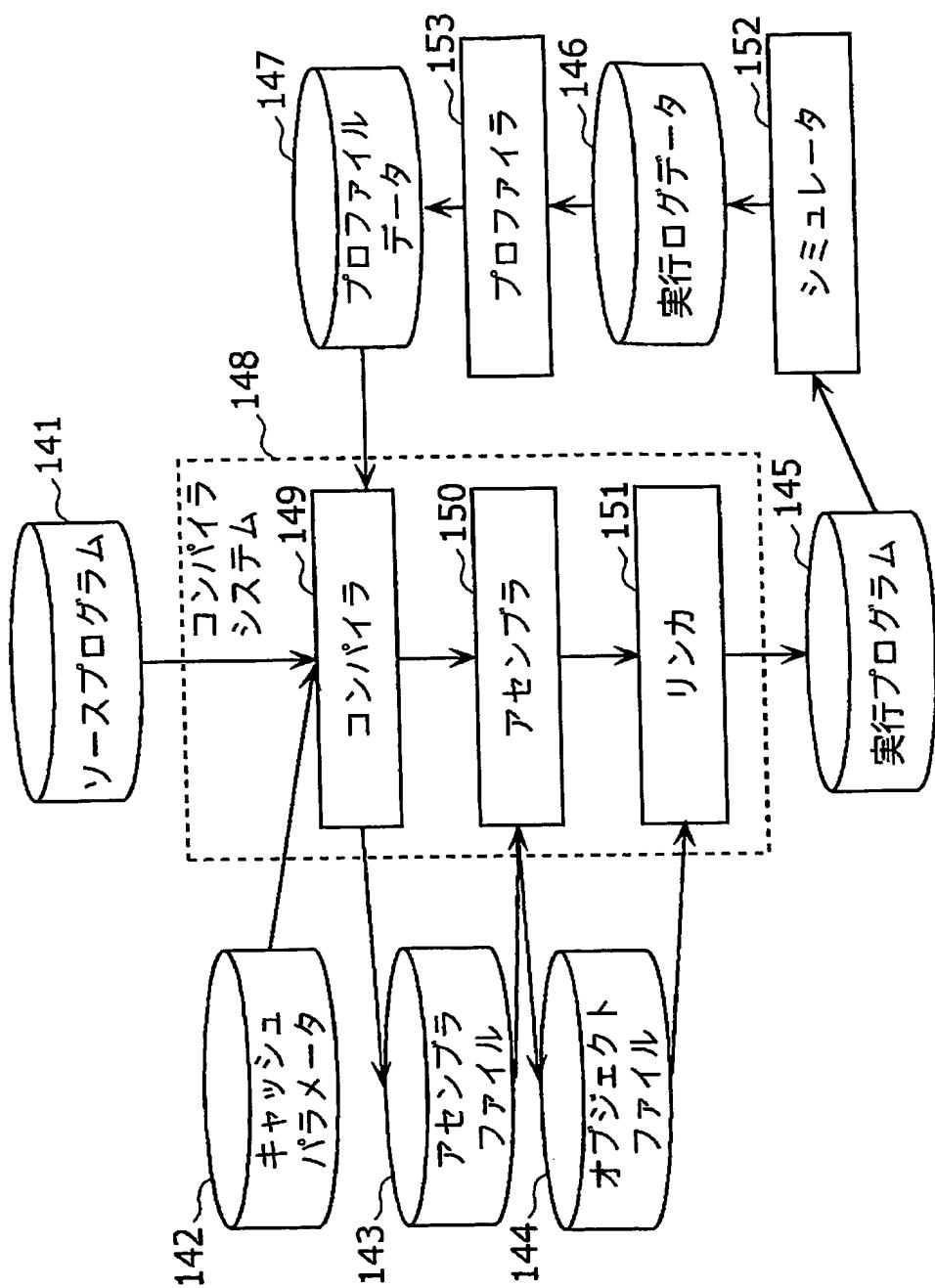
- 141 ソースプログラム
- 142 キャッシュパラメータ
- 143 アセンブラーファイル
- 144 オブジェクトファイル
- 145 実行プログラム
- 146 実行ログデータ
- 147 プロファイルデータ
- 148 コンパイラシステム
- 149 コンパイラ
- 150 アセンブラー
- 151 リンカ
- 152 シミュレータ
- 153 プロファイル
- 181 最適化補助情報
- 182 構文解析部
- 183 最適化情報解析部
- 184 一般最適化部
- 185 命令スケジューリング部
- 186 ループ構造変換部
- 187 命令最適配置部
- 188 コード出力部

【書類名】 図面
【図1】

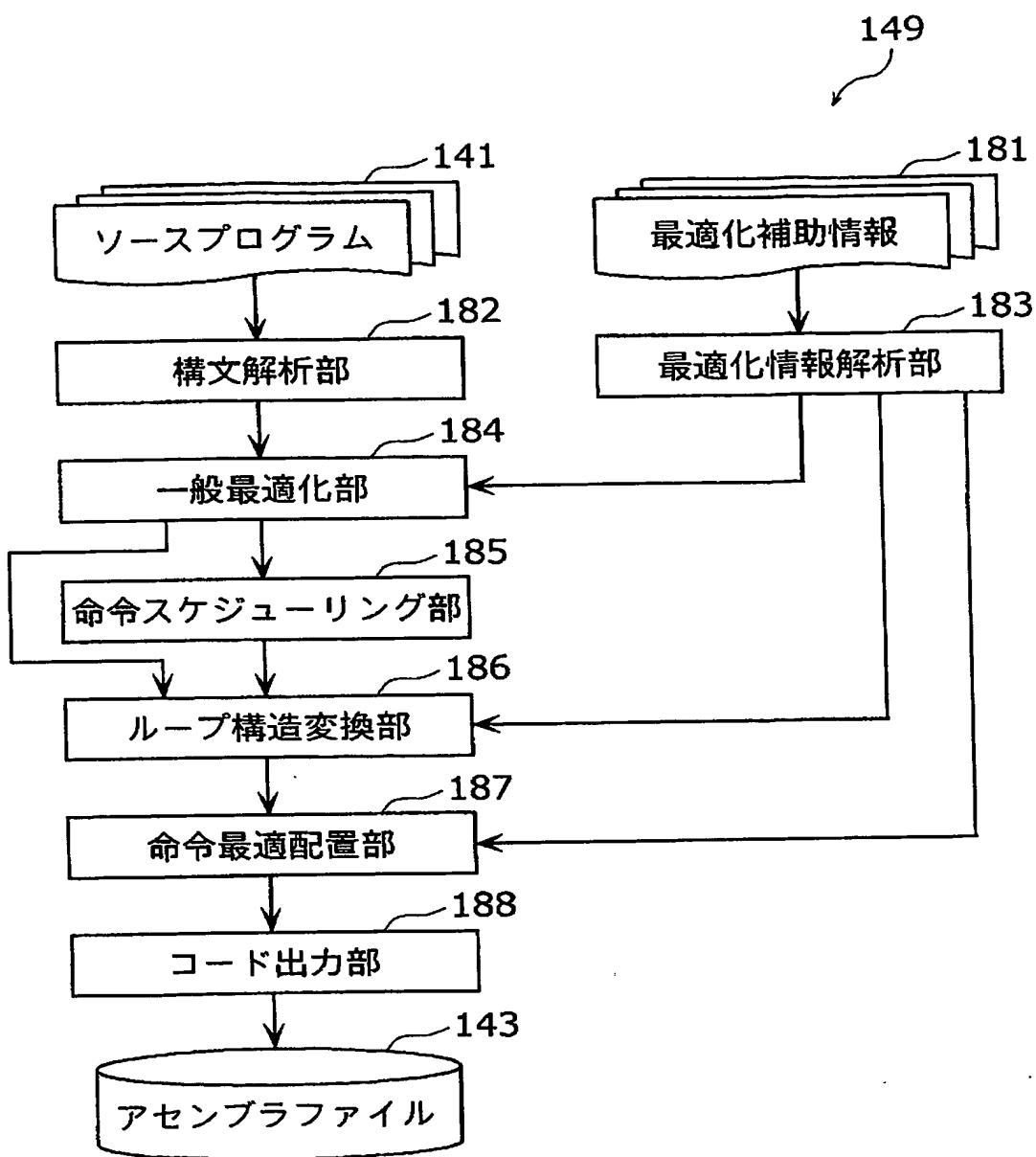
```
for (i = 0; i < 128; ) {  
    dpref(&a[i+32]);  
    for (j = 0; j < 32; j++, i++) {  
        x += a[i];  
    }  
}
```

|

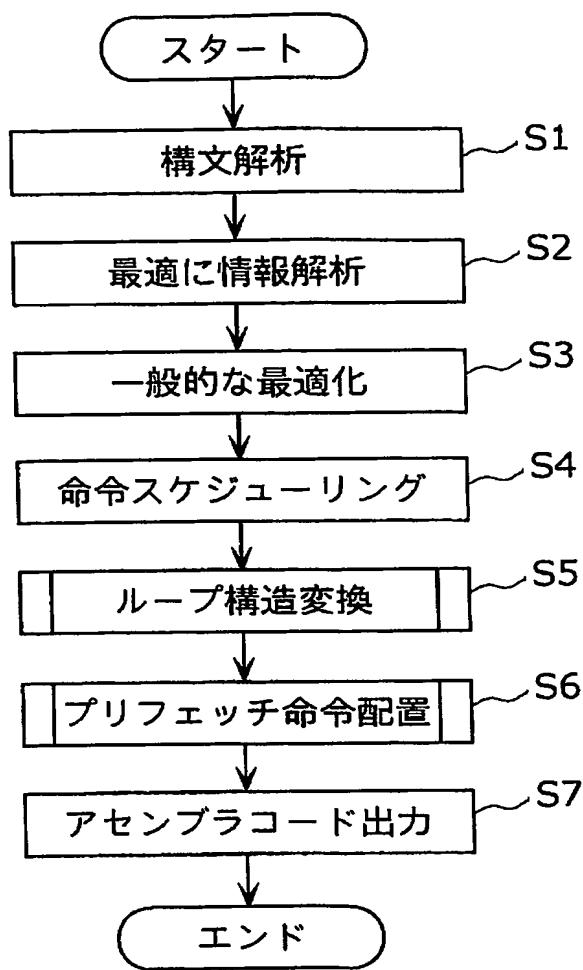
【図2】



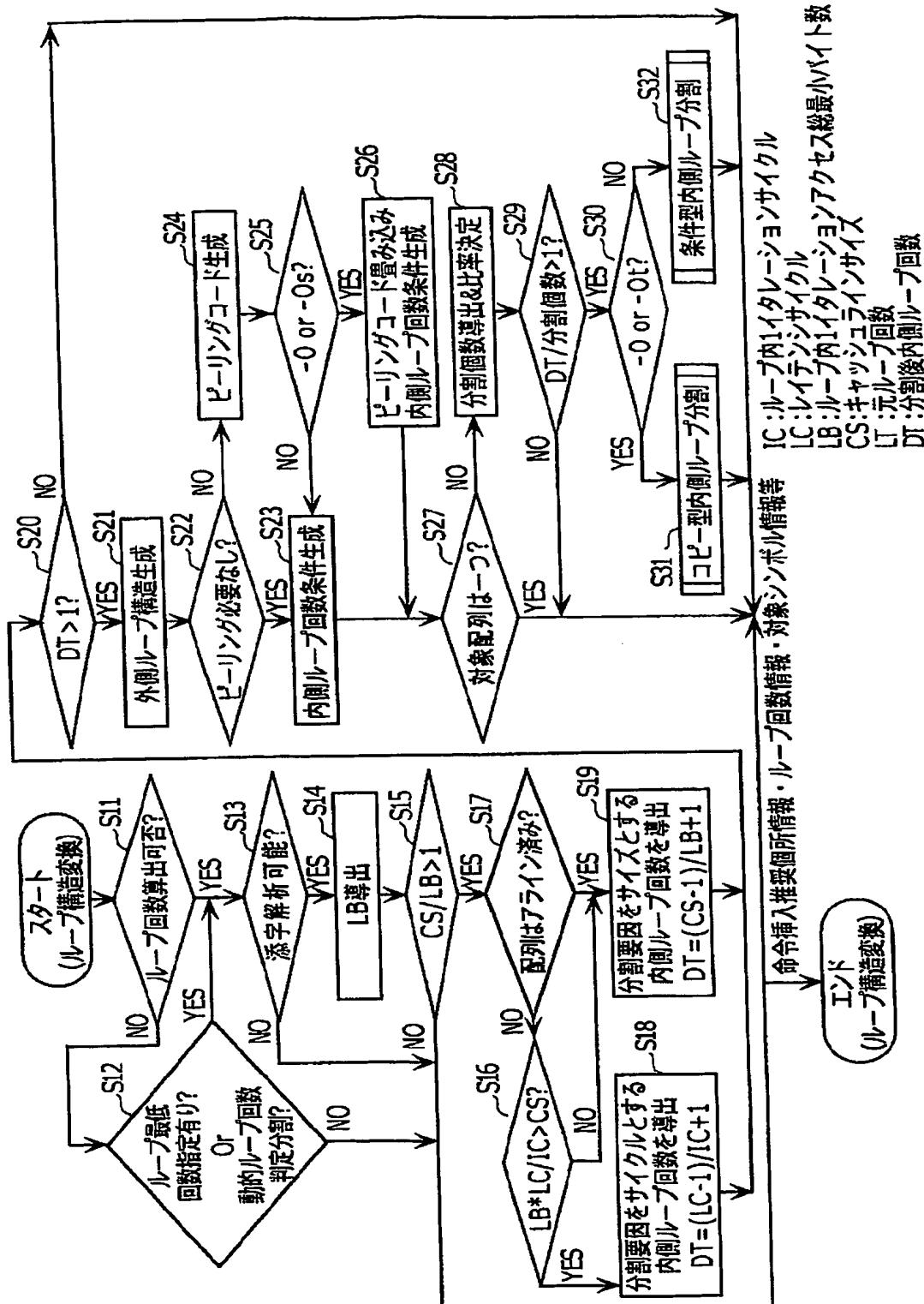
【図3】



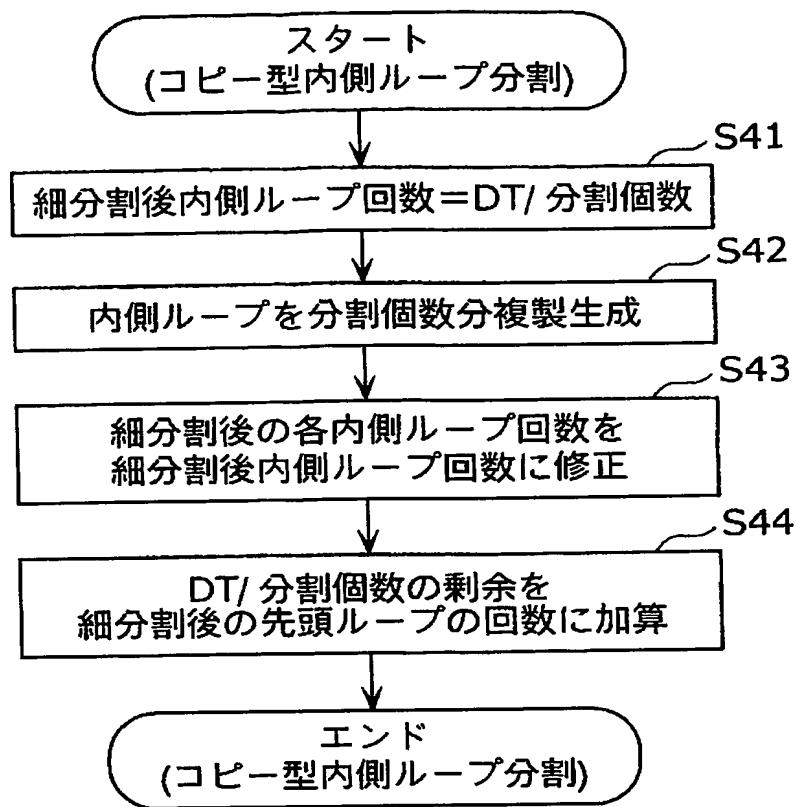
【図4】



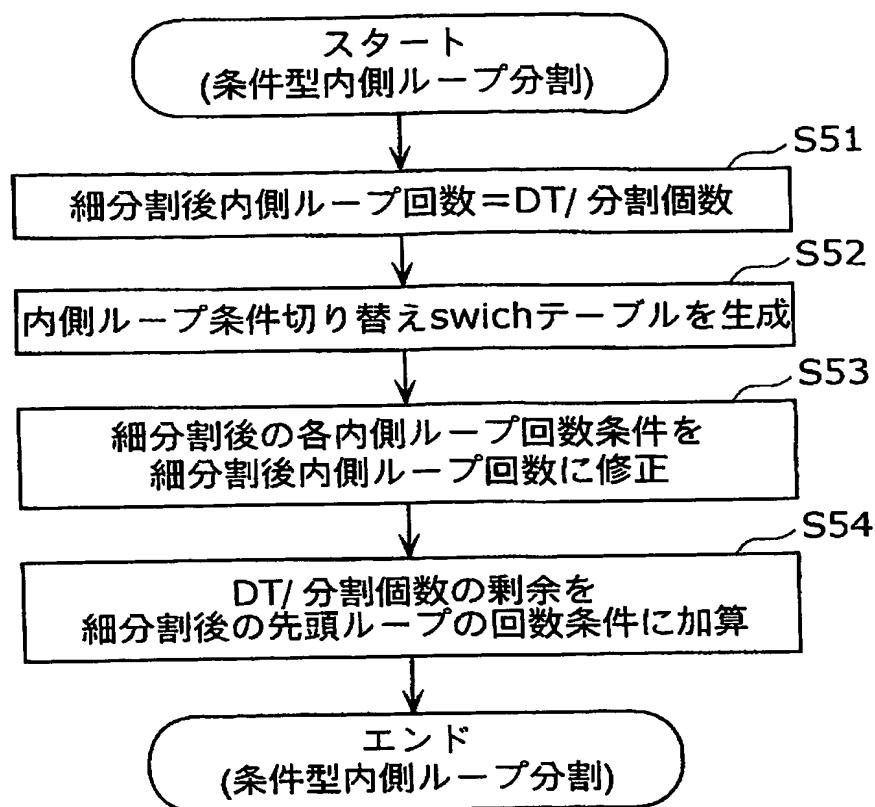
【図5】



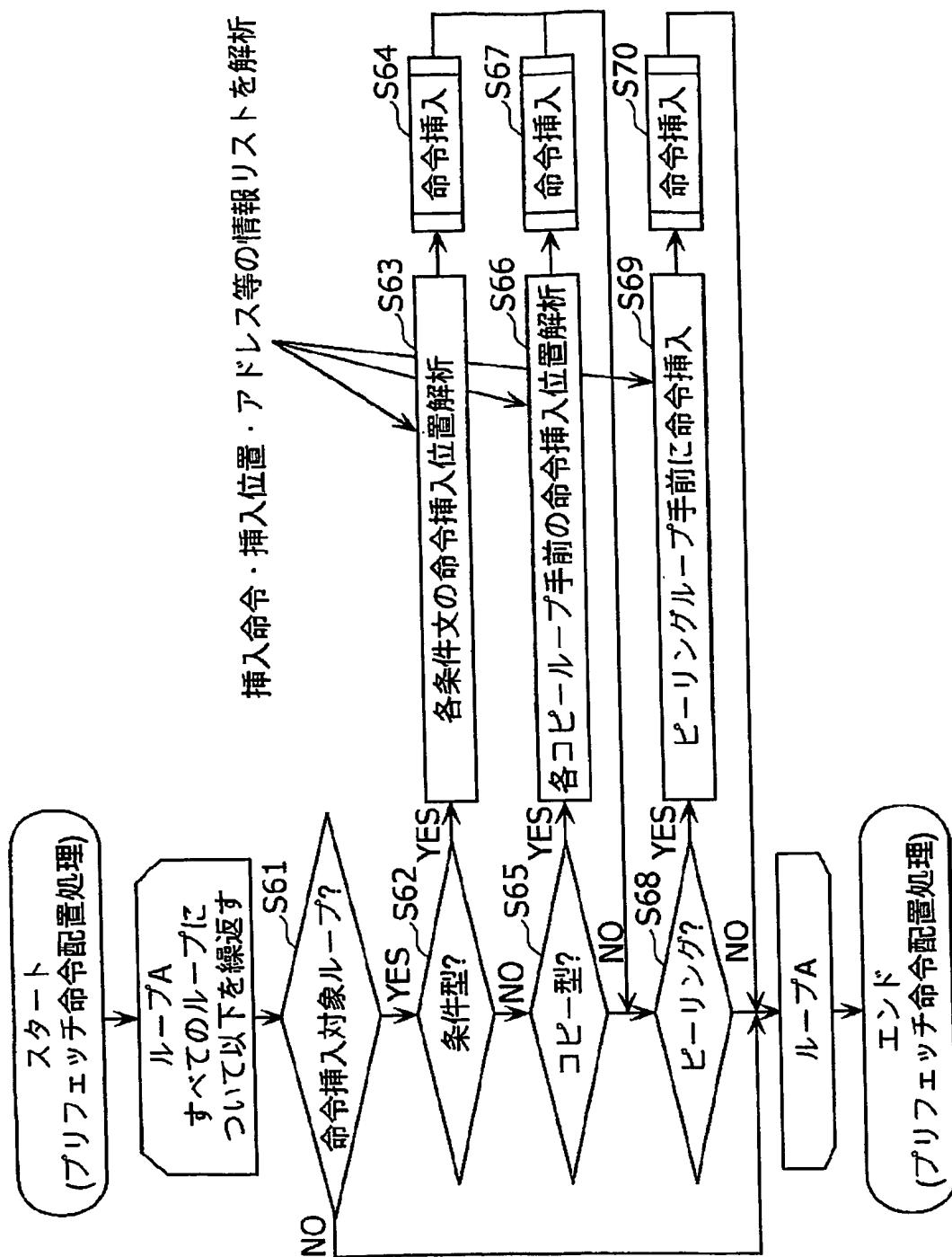
【図6】



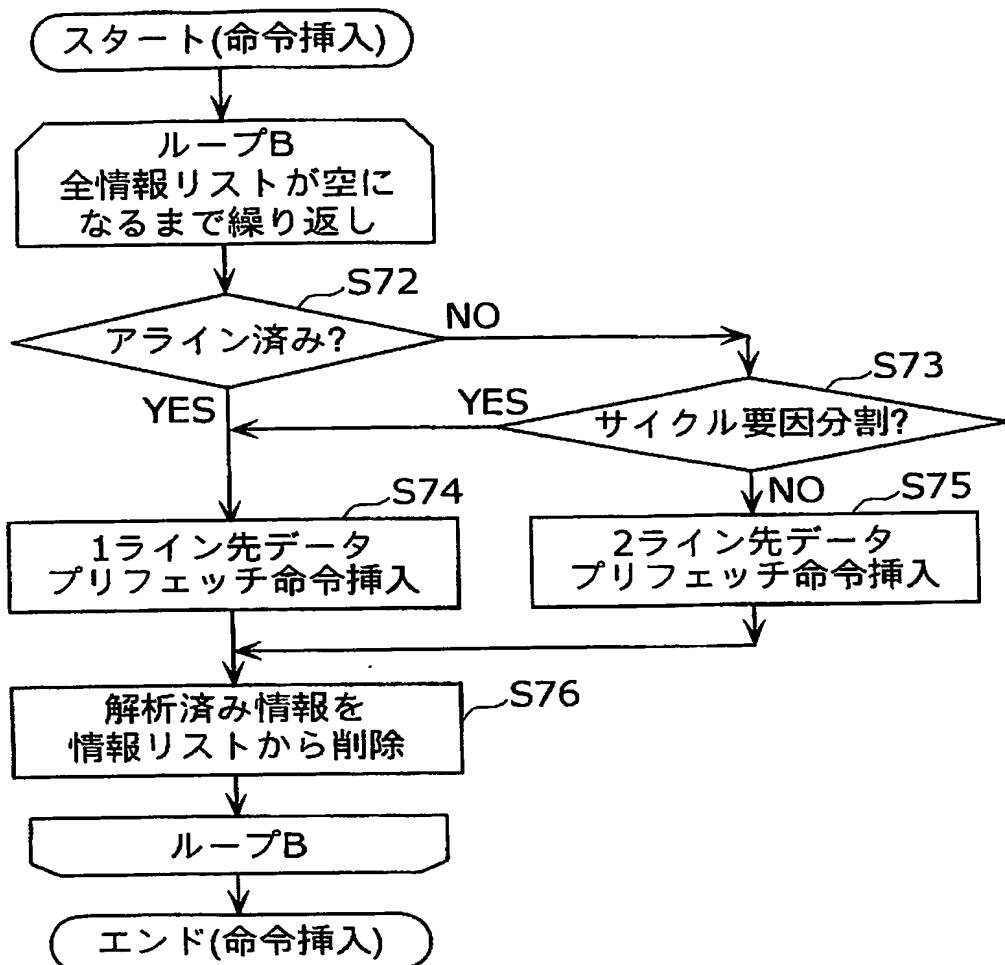
【図7】



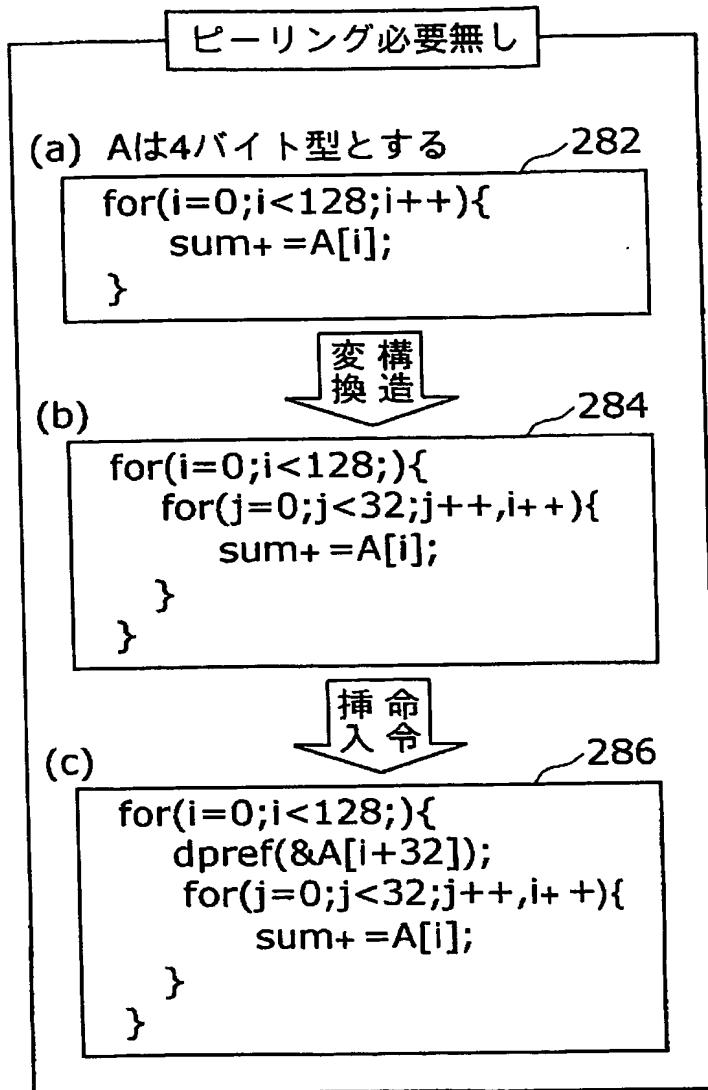
【図8】



【図9】



【図10】



【図11】

入力C言語プログラムソース 240

```
int A[1000];

int main(void)
{
    int i;
    int sum = 0;

    for ( i=0; j<128;i++ ) {
        sum += A[ i ];
    }

    return sum;
}
```

【図 12】

<pre>[PROLOG] [BGNBBLK] B1 [predecessor set] no mov REG(vr2) IMM(0) mov REG(vr5) IMM(128) ld REG(vr3) IMM(_A\$) mov REG(vr1) IMM(vr2) [ENDBBLK]</pre>	<pre>[success set] B2 REG(vr2) IMM(0) REG(vr5) IMM(128) REG(vr3) IMM(_A\$) REG(vr1) IMM(vr2)</pre>
<pre>[BGNBBLK] B2 [predecessor set] B1 B2 [label] L00001 add REG(vr2) REG(vr2), IMM(1) REG(vr4), REG(vr3) INDIRECT(vr3,0), REG(vr3), IMM(4) ldinc FLAG(C6) REG(vr2), REG(vr5) cmpit REG(vr1) REG(vr4), REG(vr1) add FLAG(C6), LAB(L00001) jmpf [ENDBBLK]</pre>	<pre>[success set] B2 B3 REG(vr2) REG(vr2), IMM(1) REG(vr4), REG(vr3) INDIRECT(vr3,0), REG(vr3), IMM(4) FLAG(C6) REG(vr2), REG(vr5) REG(vr1) REG(vr4), REG(vr1) FLAG(C6), LAB(L00001)</pre>
<pre>[BGNBBLK] B3 [predecessor set] B2 mov REG(r0) REG(vr1) ret [ENDBBLK] [EPLOG]</pre>	<pre>[success set] no REG(r0) REG(vr1)</pre>

【図13】

[PROLOG] [BGNBBLK]	B1	[predecess set] no mov REG (vr2) IMM(0) mov REG (vr5) IMM(32) ld REG (vr3) MEM(_A\$) mov REG (vr1) REG(vr2) mov REG (vr7) IMM(128)	[success set] B4
[ENDBBLK] [BGNBBLK]	B4	[predecess set] B1 B4	[success set] B2
[label]	L00002		
[ENDBBLK] [BGNBBLK]	B2	[predecess set] B4 B2	[success set] B2 B5
[label]	L00001	add REG (vr2) REG (vr1), IMM(1) ldinc REG (vr4), REG(vr3) INDIRECT(vr3,0), REG(vr3), IMM(4) cmplt FLAG(C6) REG (vr2), REG (vr5) add REG (vr1) REG (vr4), REG (vr1) jmpf FLAG (C6), LAB(L00001)	
[ENDBBLK] [BGNBBLK]	B5	[predecess set] B2 cmplt FLAG (C6) REG (vr2), REG (vr7) jmpf FLAG (C6), LAB(L00002)	[success set] B5 B3
[ENDBBLK] [BGNBBLK]	B3	[predecess set] B5 mov REG (r0) REG (vr1) ret	[success set] no
[ENDBBLK] [EPILOG]			

変換後中間言語

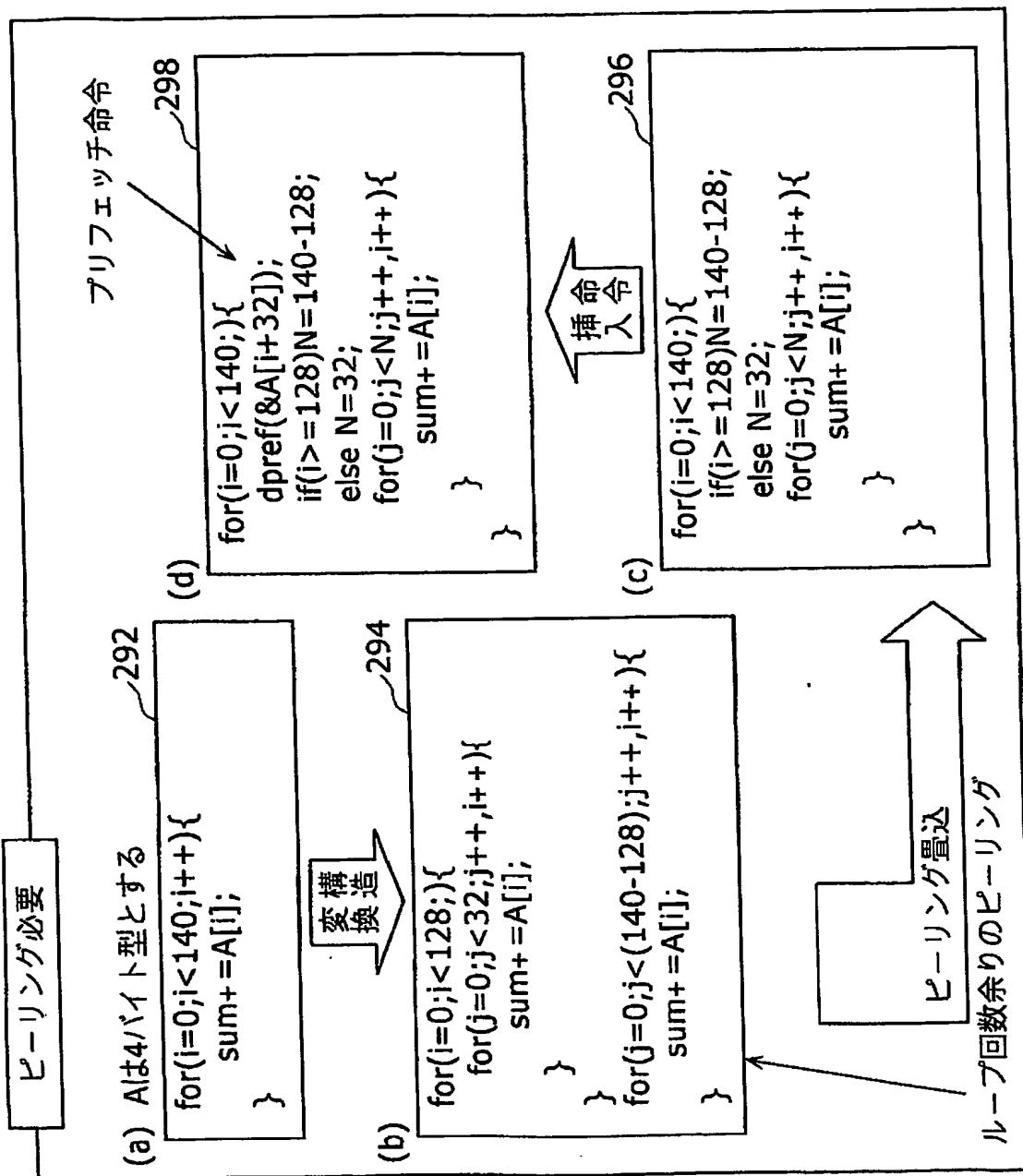
【図 14】

命令挿入後中間言語

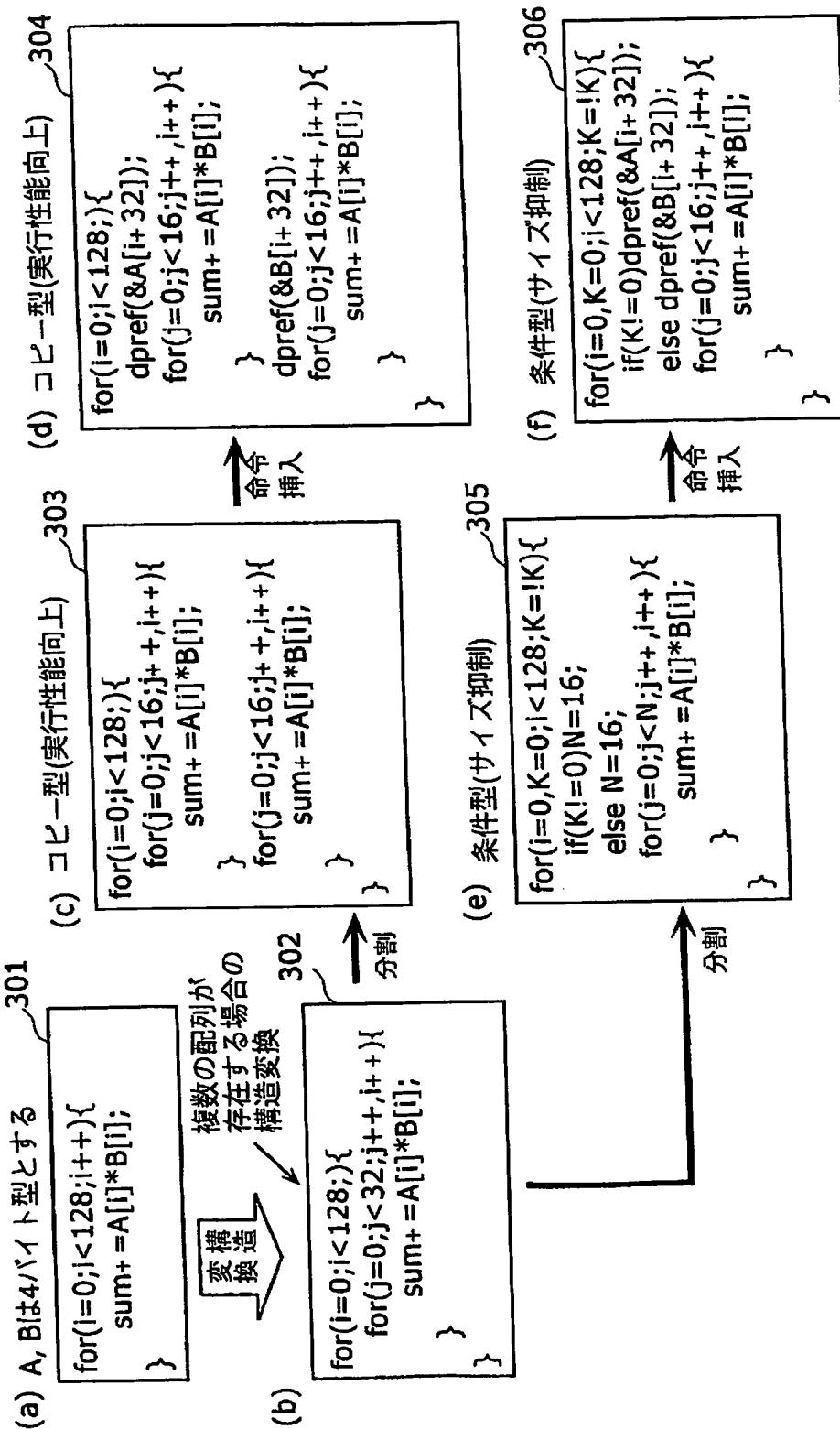
270

[PROLOG]		[success set] B4
[BGNBBLK]	B1	[predecessor set] no
mov		REG (vr2) IMM(0)
mov		REG (vr5) IMM(32)
ld		REG (vr3) MEM(_A\$)
mov		REG (vr1) REG(vr2)
mov		REG (vr7) IMM(128)
[ENDBBLK]		
[BGNBBLK]	B4	[predecessor set] B1 B4
[label] L00002		[success set] B2
dpref		INDIRECT(vr2, 128), REG (vr2)
[ENDBBLK]		
[BGNBBLK]	B2	[predecessor set] B4 B2
[label] L00001		[success set] B2 B5
add		REG (vr2) REG (vr2), IMM(1)
ldinc		REG (vr4), REG(vr3) INDIRECT(vr3,0), REG(vr3), IMM(4)
cmpit		FLAG(C6) REG (vr2), REG (vr5)
add		REG (vr1) REG (vr4), REG (vr1)
jmpf		FLAG (C6), LAB(L00001)
[ENDBBLK]		
[BGNBBLK]	B5	[predecessor set] B2 B3
cmpit		FLAG (C6) REG (vr2), REG (vr7)
jmpf		FLAG (C6), LAB(L00002)
[ENDBBLK]		
[BGNBBLK]	B3	[predecessor set] B5
mov		[success set] no
ret		REG (r0) REG (vr1)
[ENDBBLK]		
[EPILOG]		

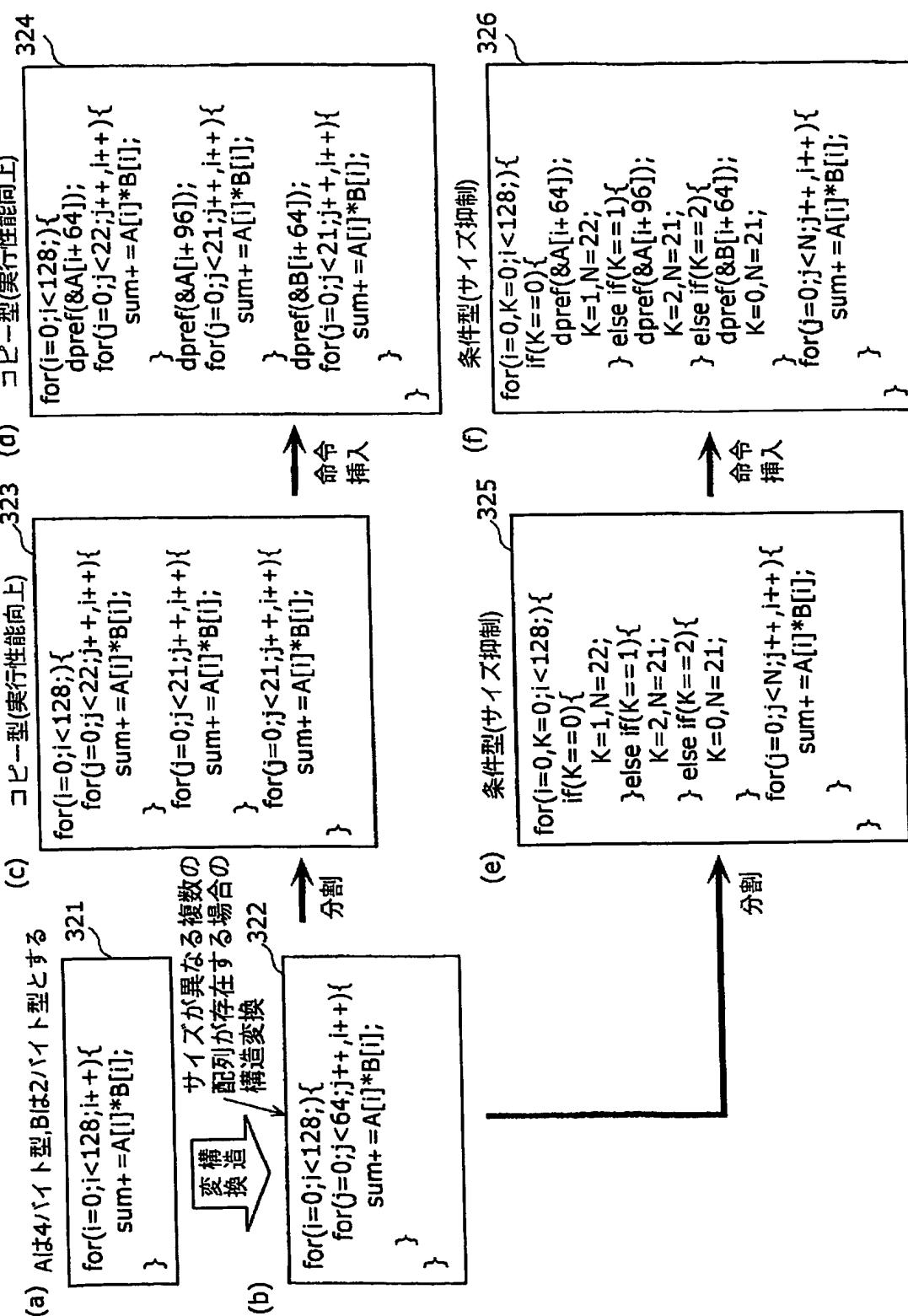
【図 15】



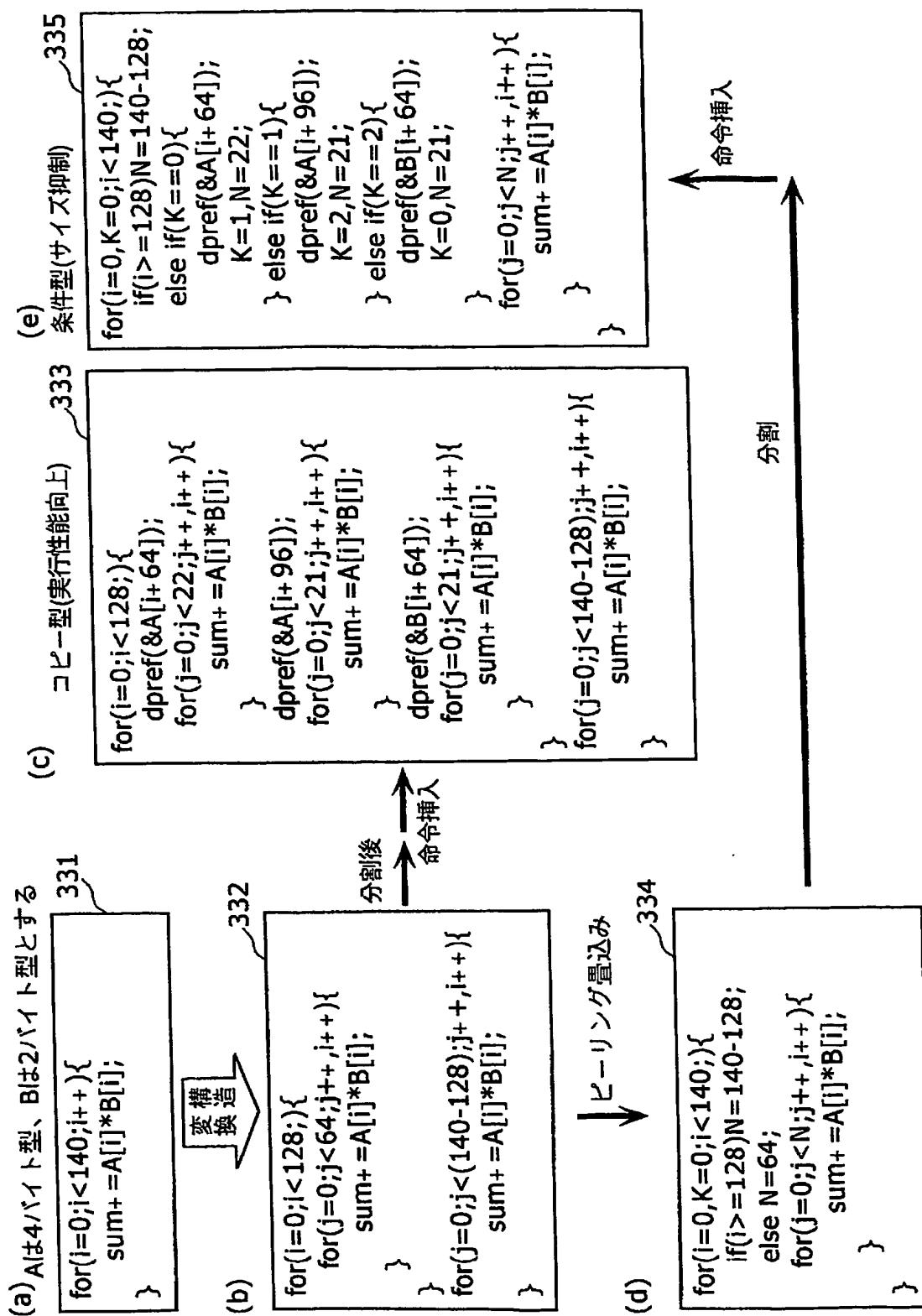
【図16】



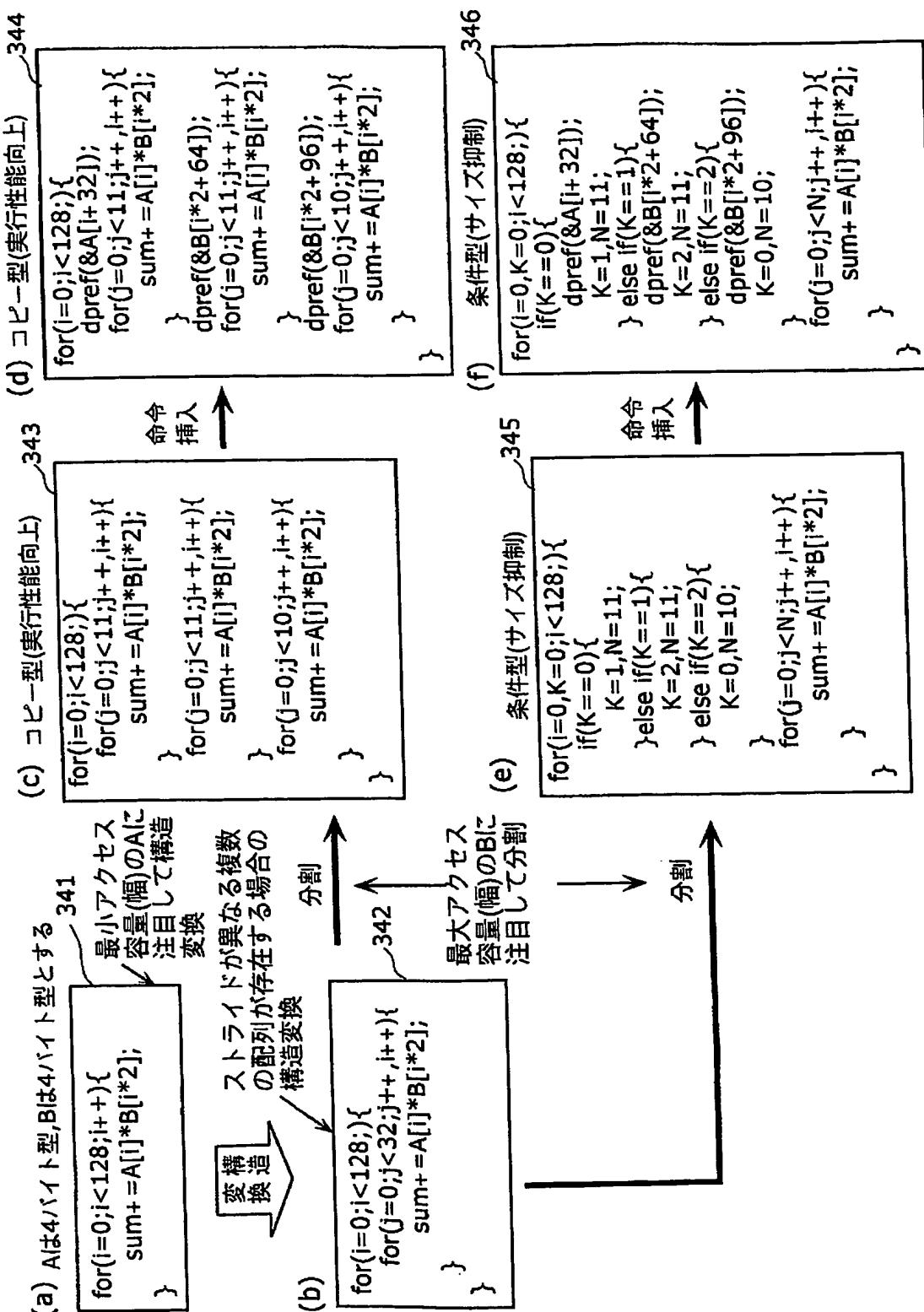
【図18】



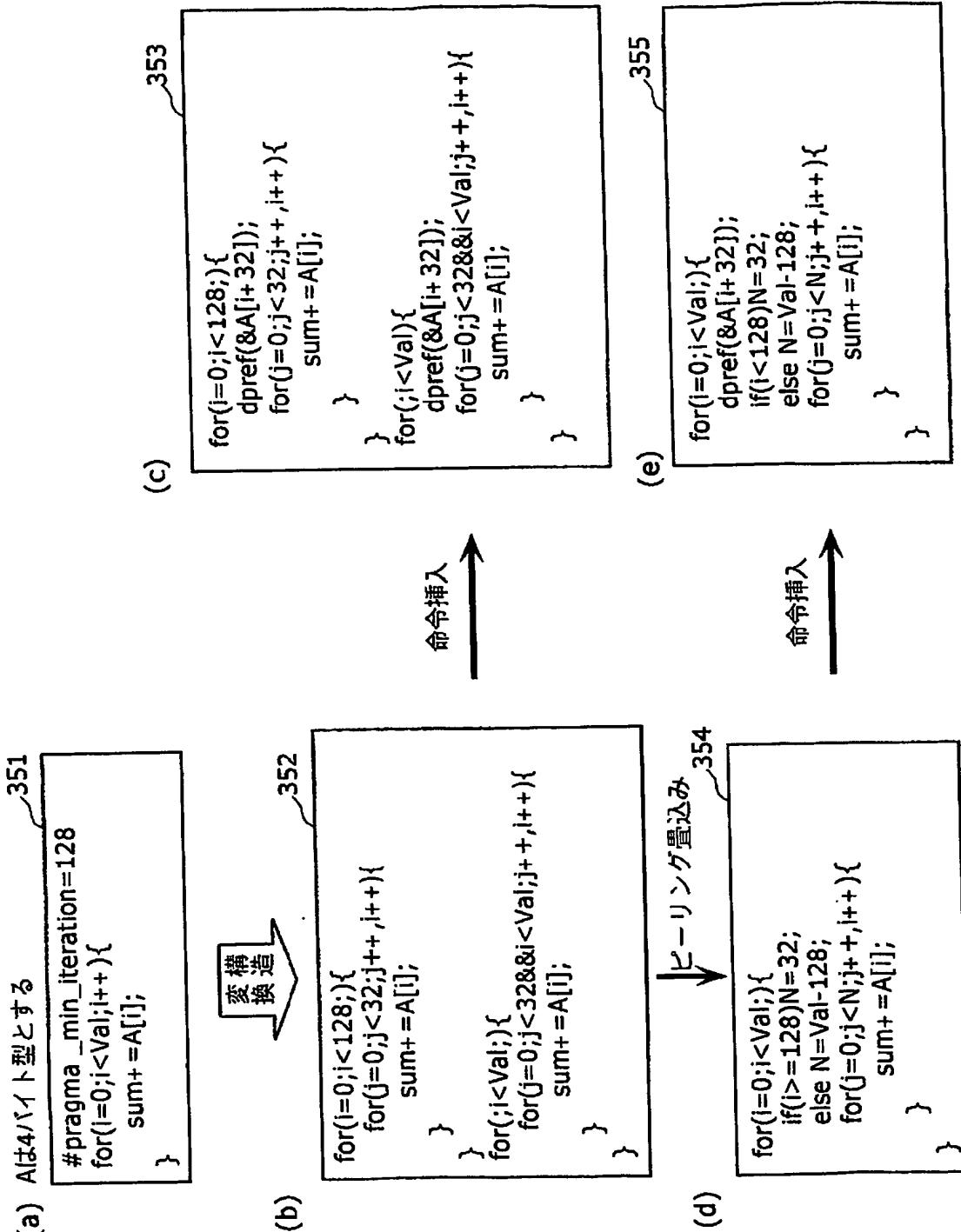
【図19】



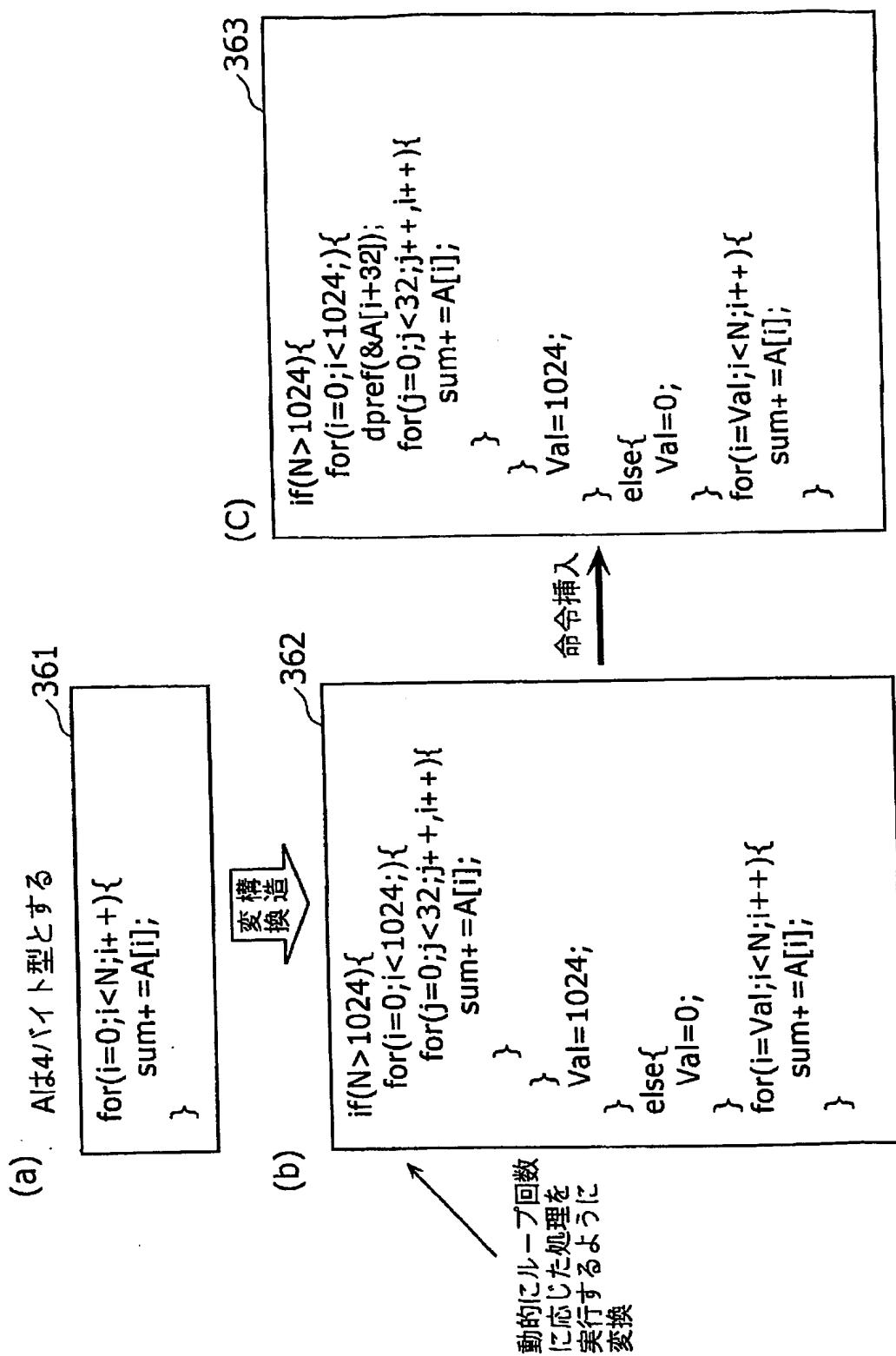
【図20】



【図21】



【図22】



【図23】

(a) Aは4バイト型とする

```

for(i=0;i<N;i++){
    sum+=A[i];
    sum+=A[i+1];
    sum+=A[i+2];
    ~省略~
    sum+=A[i+30];
    sum+=A[i+31];
}

```

371

ループ構造変換不要と
判定される場合構造
変換しないで命令挿入

(b)

↓ 命令挿入

372

```

for(i=0;i<N;i++){
    dpref(&A[i+32]);
    sum+=A[i];
    sum+=A[i+1];
    sum+=A[i+2];
    ~省略~
    sum+=A[i+30];
    sum+=A[i+31];
}

```

【図 24】

(a) Aは4バイト型とする

381

```
for(i=0;i<128;i++){
    sum+=A[i];
}
```

(b)



通常の変換を行なった後
命令挿入(2ライン先)

アライン時と
同等の変換

```
dpref(&A[0]);
dpref(&A[32]);
⋮
for(i=0;i<128;){
    dpref(&A[i+64]);
    for(j=0;j<32;j++,i++){
        sum+=A[i];
    }
}
```

382

【図25】

(a)

```
for(i=0;i<140;i++){
    sum+=A[i];
}
```

391

通常の変換を行なった後
命令挿入(2ライン先)

変換

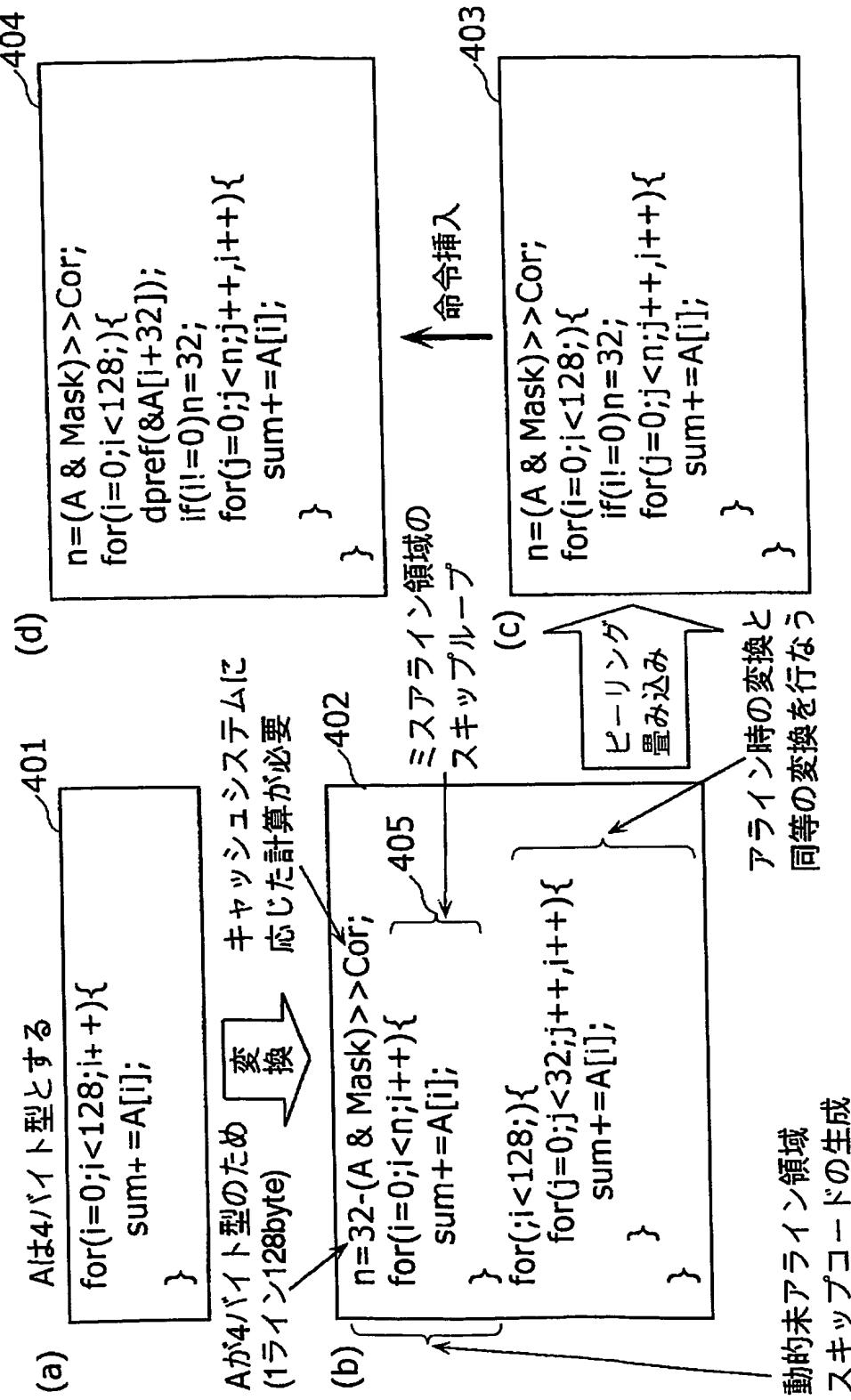
(b)

```
dpref(&A[0]);
dpref(&A[32]);
⋮
for(i=0;i<140;){
    dpref(&A[i+64]);
    if(i>=128)n=140-128;
    else n=32;
    for(j=0;j<n;j++,i++){
        sum+=A[i];
    }
}
```

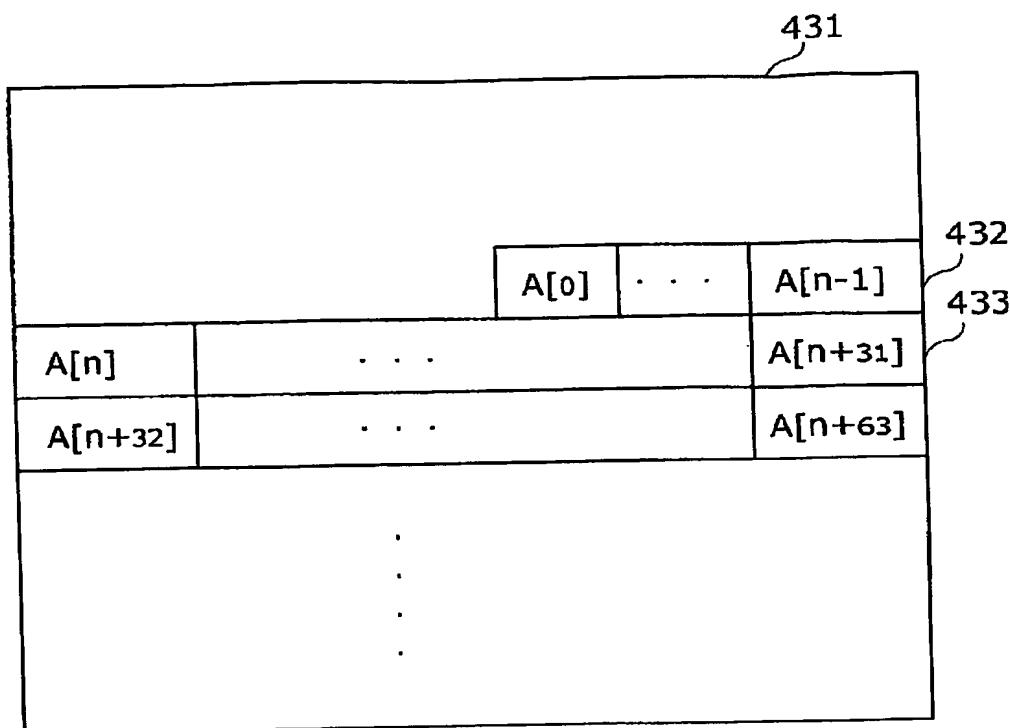
392

アライン時と
同等の変換

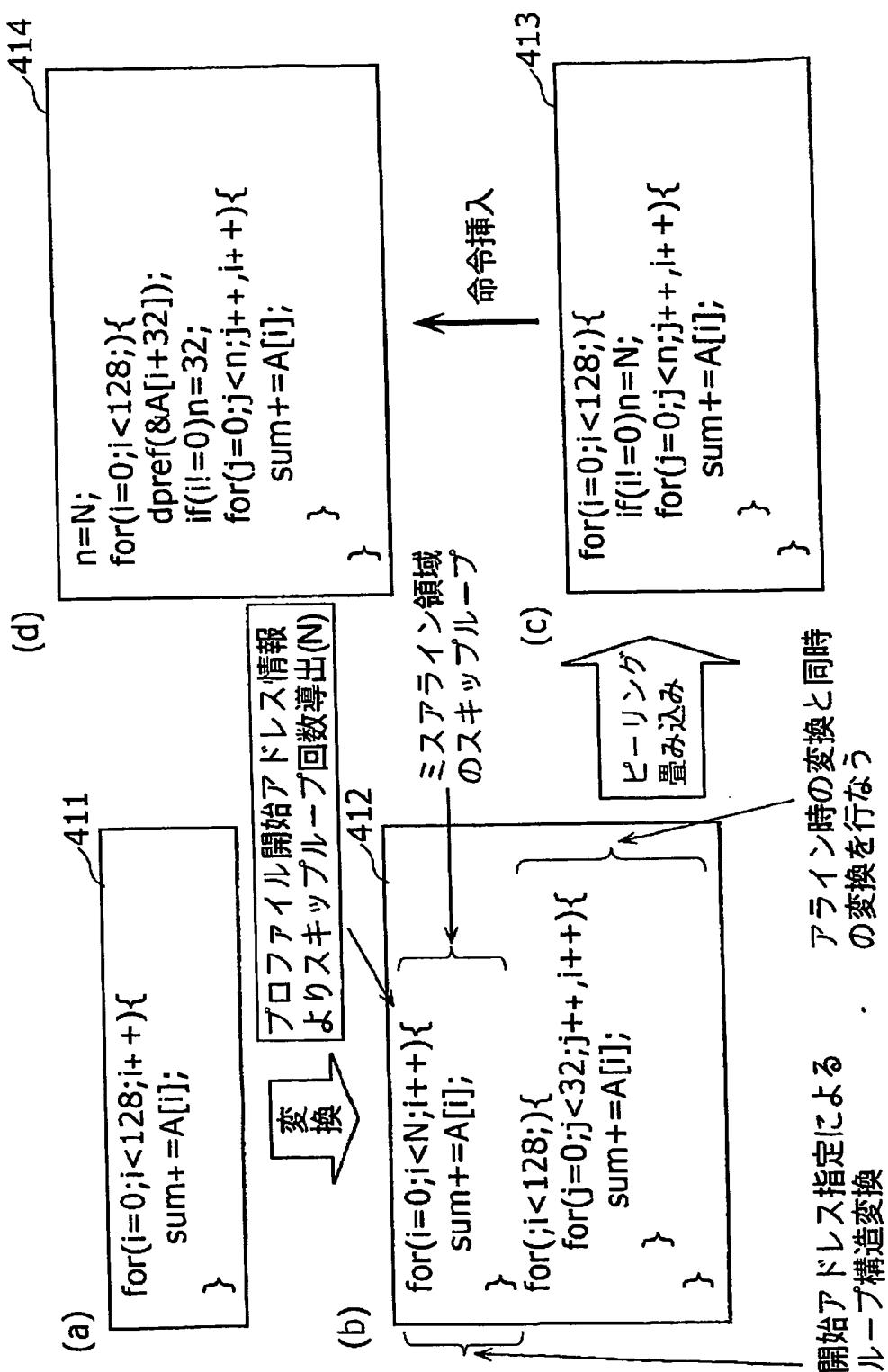
【図26】



【図 27】

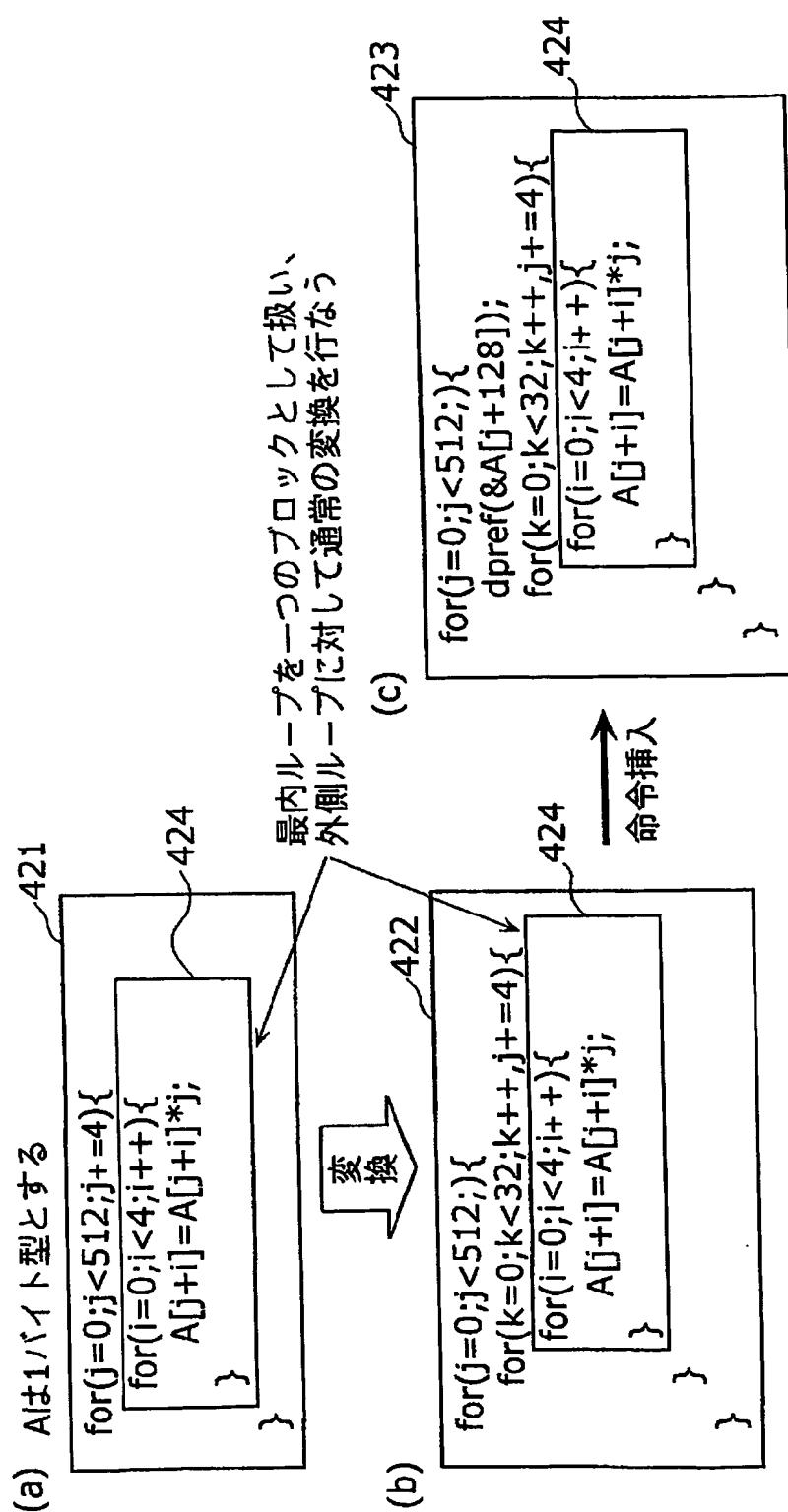


【図 28】



開始アドレス指定による アライイン時の変換と同時
ループ構造変換 . の変換を行なう

【図29】



【図30】

(a)

```
int b[128];
#pragma _loop_tiling_dpref b
for (i=0; i<128; i++)
{
    a[i] = b[i];
}
```

(b)

```
for (i=0; i<128; )
{
    dpref(&b[i+32]);
    for (j =0; j<32; j++, i++) {
        a[i] = b[i];
    }
}
```

【図31】

(a)

```
int a[];
for (i=0; i<128; i++){
    x += a[i];
}
    . . .
```

(b)

```
int a[];
for (i=0; i<128; i++){
    dpref(&a[i + N]); // 参照までのレイテンシを考慮し
    x += a[i];          // 数(ここではN)イタレーション先のデータをプリフェッチ
}
    . . .
```

【書類名】要約書

【要約】

【課題】 インターロックを引き起こす可能性のある命令を無駄に発行せずに、プログラム実行時の処理速度を向上させるコンパイラを提供する。

【解決手段】 実行時にインターロックを起こす可能性のある命令を備えたプロセッサ向けのコンパイラであって、入力プログラムに対し、ループ回数が x 回のループをループ回数が y 回のループに分割し、前記ループ回数が y 回のループを内側ループとし、ループ回数が x / y 回のループを外側ループとする二重ループ変換を行うループ構造変換部 186 と、前記二重ループ変換後のプログラムに対して、インターロックを起こす可能性のある命令の配置を行う命令最適配置部 187 としてコンピュータを機能させることを特徴とする。

【選択図】 図 3

認定・付加情報

特許出願の番号	特願 2004-035430
受付番号	50400227055
書類名	特許願
担当官	第七担当上席 0096
作成日	平成16年 2月13日

<認定情報・付加情報>

【提出日】	平成16年 2月12日
-------	-------------

特願 2004-035430

出願人履歴情報

識別番号

[000005821]

1. 変更年月日

1990年 8月28日

[変更理由]

新規登録

住 所

大阪府門真市大字門真1006番地

氏 名

松下電器産業株式会社

Document made available under the Patent Cooperation Treaty (PCT)

International application number: PCT/JP05/001670

International filing date: 04 February 2005 (04.02.2005)

Document type: Certified copy of priority document

Document details: Country/Office: JP
Number: 2004-035430
Filing date: 12 February 2004 (12.02.2004)

Date of receipt at the International Bureau: 31 March 2005 (31.03.2005)

Remark: Priority document submitted or transmitted to the International Bureau in compliance with Rule 17.1(a) or (b)



World Intellectual Property Organization (WIPO) - Geneva, Switzerland
Organisation Mondiale de la Propriété Intellectuelle (OMPI) - Genève, Suisse

**This Page is Inserted by IFW Indexing and Scanning
Operations and is not part of the Official Record**

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- BLACK BORDERS**
- IMAGE CUT OFF AT TOP, BOTTOM OR SIDES**
- FADED TEXT OR DRAWING**
- BLURRED OR ILLEGIBLE TEXT OR DRAWING**
- SKEWED/SLANTED IMAGES**
- COLOR OR BLACK AND WHITE PHOTOGRAPHS**
- GRAY SCALE DOCUMENTS**
- LINES OR MARKS ON ORIGINAL DOCUMENT**
- REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY**
- OTHER:** _____

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.